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
Бородина Артёма Горовича

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 Теоретические сведения по рассмотренным темам с их обоснованием

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

Классификация свойств операций.

Бинарная операция $\cdot$ на множестве A называется:

1. **Ассоциативной**, если $\forall x, y, z \in A$ выполняется равенство $x \cdot (y \cdot z) = (x \cdot y) \cdot z$;
2. **Коммутативной**, если $\forall x, y \in A$ выполняется равенство $x \cdot y = y \cdot x$;
3. **Идемпотентной**, если $\forall x \in A$ выполняется равенство $x \cdot x = x$;
4. **Обратимой**, если $\forall x, y \in A$ уравнения $x \cdot a = y$ и $b \cdot x = y$ имеют решение, причём единственное;
5. **Дистрибутивной относительно операции +**, если $\forall x, y, z \in A$ выполняются равенства $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ и $(y + z) \cdot x = (y \cdot x) + (z \cdot x)$.

Действия над бинарными отношениями.

Определим следующие действия над бинарными отношениями:

1. **Пересечением** $P \cap Q$ отношений $P \subset A \times B$ и $Q \subset A \times B$ называется отношение, которое содержит только общие для P и Q пары: $P \cap Q = \{(x, y) : (x, y) \in P \text{ и } (x, y) \in Q\}$;
2. **Объединением** $P \cup Q$ отношений $P \subset A \times B$ и $Q \subset A \times B$ называется отношение, которое включает все пары, содержащиеся или в подмножестве P или в подмножестве Q : $P \cup Q = \{(x, y) : (x, y) \in P \text{ или } (x, y) \in Q\}$. Когда объединение $P \cup Q$ содержит все возможные пары из $A \times B$, а пересечение P и Q пусто, то говорят, что отношения P и Q **образуют разбиение** $A \times B$, а их объединение есть **полное отношение**;
3. **Дополнением** $\overline{P} \subset A \times B$ отношения $P \subset A \times B$ называется отношение, состоящее из тех пар $(x, y) \in A \times B$, которые не входят в P : $\{(x, y) : (x, y) \in A \times B \text{ и } (x, y) \notin P\}$. Отношения P и $\overline{P}$ образуют **разбиение** $A \times B$, т.е. $P \cup \overline{P} = A \times B$ и $P \cap \overline{P} = \varnothing$.
4. **Обратным отношением** $P^{-1} \subset B \times A$ к отношению $P \subset A \times B$ называется отношение, которое содержит пару (x, y) тогда и только тогда, когда $(y, x) \in P$, т.е. $P^{-1} = \{(x, y) : (y, x) \in P\}$;

5. **Композицией (произведением)** $P \circ Q$ отношений $P \subset A \times B$ и $Q \subset B \times C$ называется отношение, которое содержит пару (x, y) тогда и только тогда, когда существует $z \in B$ такое, что $(x, z) \in P$ и $(z, y) \in Q$, т.е. $P \circ Q = \{(x, y) :$ найдётся z такое, что $(x, z) \in P$ и $(z, y) \in Q\}$. К частным случаям композиции относится **квадрат отношения** $P : P^2 = \{(x, y) :$ найдётся z такое, что $(x, z) \in P$ и $(z, y) \in P\}$. По индукции определяется n-ая **степень отношения** $P : P^n = P^{n-1} \circ P$.

Проверка ассоциативности операции по тесту Лайта.

В случае конечного n-элементного множества $S = \{s_1, s_2, \ldots, s_n\}$ операция умножения на S задаётся *таблицей Кэли* размерности $n \times n$, строки и столбцы которой помечены элементами множества S и в которой на пересечении i-ой строки и j-го столбца стоит произведение $s_i \cdot s_j$ элементов s_i, s_j.

$\cdot$	s_1	s_2	$\ldots$	s_n
s_1	$s_1 \cdot s_1$	$s_1 \cdot s_2$	$\ldots$	$s_1 \cdot s_n$
s_2	$s_2 \cdot s_1$	$s_2 \cdot s_2$	$\ldots$	$s_2 \cdot s_n$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
s_n	$s_n \cdot s_1$	$s_n \cdot s_2$	$\ldots$	$s_n \cdot s_n$

Ассоциативность такой операции определяется по *тесту Лайта*: для доказательства тождества ассоциативности последовательно фиксируем элементы $a \in S$ и проверяем выполнимость равенства: $(x \cdot a) \cdot z = x \cdot (a \cdot z)$ для любых $x, z \in S$ путём сравнения следующих двух таблиц, составленных для произведений соответственно левой и правой частей этого равенства.

Таблица произведений $(x \cdot a) \cdot z$ левой части равенства:

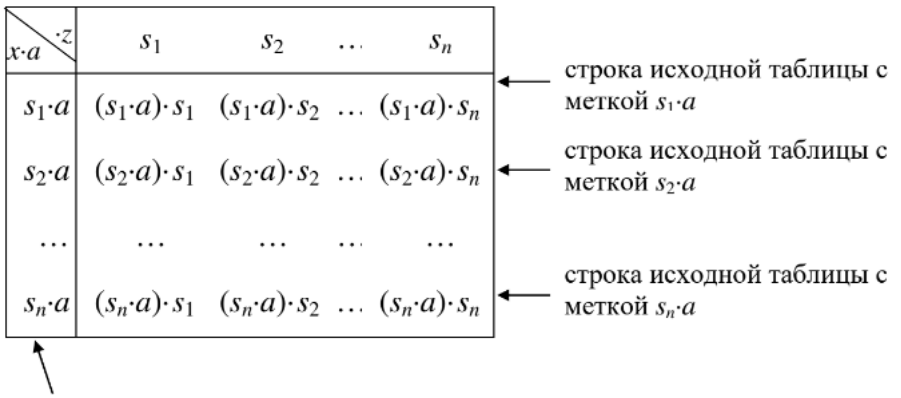
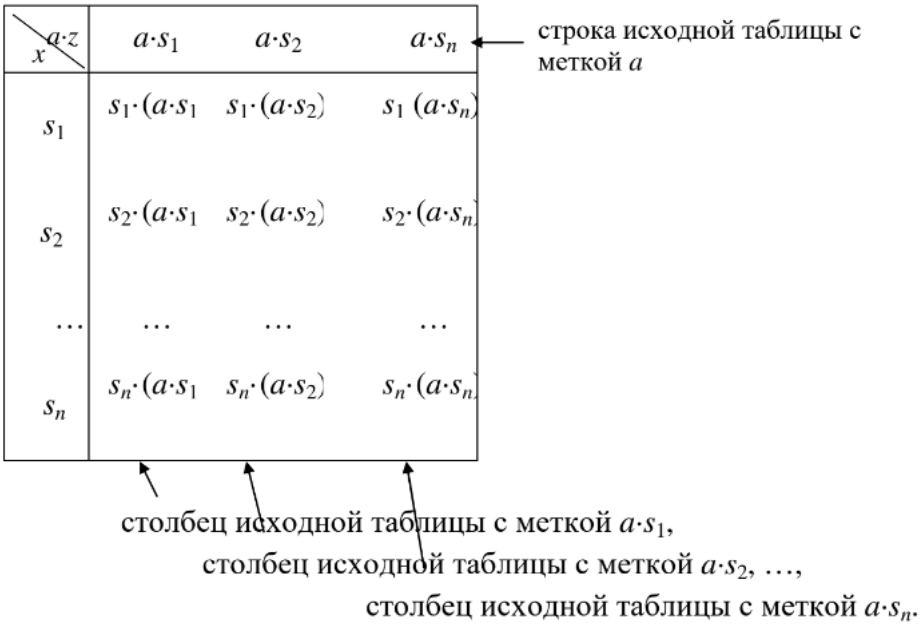


Таблица произведений $x \cdot (a \cdot z)$ правой части равенства:



3 Результаты работы

3.1 Алгоритм проверки свойства ассоциативности операции

Описание алгоритма проверки свойства ассоциативности операции.

Вход: множество объектов S и таблица Кэли рассматриваемой операции $\cdot$ над множеством объектов.

Выход: сообщение 'Операция является ассоциативной.' или 'Операция не является ассоциативной.'

Метод: Ассоциативность операции определяется по *тесту Лайта*: для доказательства тождества ассоциативности последовательно фиксируем элементы $a \in S$ (S – конечное n-элементное множество объектов) и проверяем выполнимость равенства: $(x \cdot a) \cdot z = x \cdot (a \cdot z)$ для любых $x, z \in S$ путем сравнения таблиц произведений $(x \cdot a) \cdot z$ левой части равенства и $x \cdot (a \cdot z)$ правой части равенства.

Псевдокод алгоритма проверки свойства ассоциативности операции.

```
check_associativity(<N\matrix N> CayleyTable, objectSet[])
{
  pair<<matrix>, <matrix>> tables;
  for object in objectSet:
  {
    tables = construct_tables(CayleyTable, object);
    if (tables.left != tables.right)
      return (false);
  }
  return (true);
}
```

Листинг 1: Псевдокод алгоритма.

Код программы проверки свойства ассоциативности операции.

```
pair<vector<vector<int>>, vector<vector<int>>> constructTables (
  vector<vector<int>> CayleyTable, int object)
{
  int i, j;
  vector<vector<int>> leftTable (
    matrixDimension, vector<int>(matrixDimension, 0)),
                     rightTable (
   matrixDimension, vector<int>(matrixDimension, 0));

```

```
  for (i = 0; i < matrixDimension; ++i)
    for (j = 0; j < matrixDimension; ++j)
      {
        leftTable[i][j] =
          CayleyTable[CayleyTable[i][object - 1] - 1][j];
        rightTable[i][j] =
          CayleyTable[i][CayleyTable[object - 1][j] - 1];
      }
  return (make_pair (leftTable, rightTable));
}

bool checkAssociativity (vector<vector<int>> CayleyTable)
{
  int i;
  pair<vector<vector<int>>, vector<vector<int>>> tables;

  for (i = 0; i < objectSet.size (); ++i)
    {
      tables = constructTables (CayleyTable, objectSet[i]);
      if (tables.first != tables.second)
        return (false);
    }
  return (true);
}
```

Листинг 2: Код программы.

Оценка временной сложности алгоритма проверки свойства ассоциативности операции.

Алгоритм был реализован наивно – он заключается в проверке всех возможных троек аргументов путём построения и сравнения элементов левой и правой таблиц произведений. Такая реализация требует $O(n^3)$ времени, где n – размер множества объектов, над которым определена операция.

3.2 Алгоритм проверки свойства коммутативности операции

Описание алгоритма проверки свойства коммутативности операции.

Вход: множество объектов S и таблица Кэли $CayleyTable$ рассматриваемой операции $\cdot$ над множеством объектов.

Выход: сообщение 'Операция является коммутативной.' или 'Операция не является коммутативной.'

Метод: проверяются только элементы, для которых $j \geq i$ ($i = 1, \ldots, N,\ j = 1, \ldots, N,\ N$ – размер множества объектов) – элементы, расположенные над главной диагональю в таблице Кэли. Между собой проверяются значения элементов $CayleyTable[i][j]$ и $CayleyTable[j][i]$. Если все пары рассмотренных элементов равны между собой, то операция коммутативна, иначе некоммутативна.

Псевдокод алгоритма проверки свойства коммутативности операции.

```
check_commutativity (<N\timesN matrix> CayleyTable)
{
  for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
      if (CayleyTable[i][j] != CayleyTable[j][i])
        return false;
  return true;
}
```

Листинг 3: Псевдокод алгоритма.

Код программы проверки свойства коммутативности операции.

```
bool checkCommutativity (vector<vector<int>> CayleyTable)
{
  int i, j;

  for (i = 0; i < CayleyTable.size(); ++i)
    for (j = i; j < CayleyTable.size(); ++j)
      if (CayleyTable[i][j] != CayleyTable[j][i])
        return (false);
  return (true);
}
```

Листинг 4: Код программы.

Оценка временной сложности алгоритма проверки свойства коммутативности операции.

Рассмотрим таблицу Кэли $CayleyTable$ операции $\cdot$ над множеством объектов S размера n. В первой строке таблице Кэли осуществляется n проверок, во второй $n-1$ и т.д. вплоть до последней строки таблицы, где осуществляется одна проверка. Но $1+2+\cdots+n=(n^2+n)/2$. Таким образом, оценка временной сложности алгоритма проверки коммутативности операции $\cdot$ принимает вид: $O((n^2+n)/2)=O(n^2)$.

3.3 Алгоритм проверки свойства идемпотентности операции

Описание алгоритма проверки свойства идемпотентности операции.

Вход: множество объектов S и таблица Кэли $CayleyTable$ рассматриваемой операции $\cdot$ над множеством объектов.

Выход: множество идемпотентов операции $\cdot$ и сообщение 'Операция является идемпотентой.' или 'Операция не является идемпотентной.'

Метод: проверка на идемпотентность осуществляется проходом по главной диагонали таблицы Кэли. Элемент $x \in S$ называется идемпотентом операции $\cdot$, если $x \cdot x = x$. Операция называется идемпотентной, если все объекты, над которой она определена – идемпотенты этой операции. В случае идемпотентности элемента $x \in S$ он заносится в множество идемпотентов этой операции, которое затем выводится.

Псевдокод алгоритма проверки свойства идемпотентности операции.

```
check_idempotency(<N\times N matrix> CayleyTable, objectSet[])
{
  idempotents[];
  for i in range(N):
    if (CayleyTable[i][i] == objectSet[i])
      idempotents.push_back(objectSet[i]);
  print(idempotents);
  return (N == idempotents.size());
}
```

Листинг 5: Псевдокод алгоритма.

Код программы проверки свойства идемпотентности операции.

```
bool checkIdempotency (vector<vector<int>> CayleyTable)
{
  int i;
  vector<int> idempotents;

  for (i = 0; i < matrixDimension; ++i)
    if (CayleyTable[i][i] == objectSet[i])
      idempotents.push_back (objectSet[i]);

  int idSize = idempotents.size ();
  cout << "IDEMPOTENTS OF THIS OPERATION ";
  if (idSize == 0)
    cout << "IS: EMPTY SET.\n";
```

```
  else if (idSize == 1)
    cout << "IS: " << idempotents[0] << ".\n";
  else if (idSize == objectSet.size ())
    cout << "IS: OBJECT SET. ";
  else
    {
      cout << "ARE: ";
      for (i = 0; i < idSize; ++i)
        cout << idempotents[i] <<
          (i == idSize - 1 ? ". " : ", ");
    }
  return (idSize == objectSet.size ());
}
```

Листинг 6: Код программы.

Оценка временной сложности алгоритма проверки свойства идемпотентности операции.

Рассмотрим таблицу Кэли размерности $N \times N$ операции $\cdot$. Временная сложность алгоритма проверки свойства идемпотентности операции: $O(N)$, т.к. проход осуществляется только по элементам главной диагонали таблицы Кэли.

Результат тестирования программы проверки свойств заданной операции.

Рассмотрим операцию $\cdot$ со своей таблицей Кэли размерности $N \times N$. Проверим свойства ассоциативности, коммутативности и идемпотентности заданной операции $\cdot$.

```
INPUT MATRIX DIMENSION:
4
INPUT AN OBJECT SET:
1 2 3 4
INPUT MATRIX ELEMENTS:
1 2 3 4
2 3 4 1
3 4 1 2
4 1 2 3
OPERATION IS ASSOCIATIVE.
OPERATION IS COMMUTATIVE.
IDEMPOTENTS OF THIS OPERATION IS: 1.
OPERATION IS NOT IDEMPOTENT.
```

Рисунок 1 – Результат проверки свойств заданной операции $\cdot$.

Как видно, рассматриваемая нами операция $\cdot$ является коммутативной и ассоциативной, но не является идемпотентной.

3.4 Алгоритм проверки свойства дистрибутивности операции

Описание алгоритма проверки свойства дистрибутивности операции.

Вход: множество объектов S и таблицы Кэли операций $+$ и $\cdot$ над этим множеством объектов.

Выход: сообщение о том, является ли операция $\cdot$ право-, лево- или просто дистрибутивной относительно операции $+$.

Метод: сначала операция $\cdot$ проверяется на коммутативность. Если операция коммутативна и дистрибутивна, то она и леводистрибутивна, и праводистрибутивна. Если же операция не будет коммутативна, то она может оказаться как только леводистрибутивной, так и только праводистрибутивной. Для каждого элемента x из множества объектов S строятся таблицы вида $(b + c) \cdot x$ и $(b \cdot x) + (c \cdot x)$ – для проверки правой дистрибутивности и $x \cdot (b + c)$ и $(x \cdot b) + (x \cdot c)$ – для проверки левой дистрибутивности. Если операция коммутативна, то неважно, какую из таблиц строить.

Псевдокод алгоритма проверки свойства дистрибутивности операции.

```
1  check_distributivity(<N×N matrix> multCayTab,
2    <N×N matrix> addCayTab, objectSet[])
3  {
4    bool leftDistributive = rightDistributive = true;
5
6    if (check_commutativity(multCayTab))
7    {
8      for object in objectSet:
9        if construct_left_tables(
10             multCayTab, addCayTab, object) are not equal:
11         print('Distributive.');
12     print('Not distributive.');
13   }
14   else
15   {
16     for object in objectSet:
17       if construct_left_tables(
18            multCayTab, addCayTab, object) are not equal
19       {
20         leftDistributive = false;
21         print('Not left-distributive.');
```

```
        break;
      }
    if (leftDistributive)
      print('Left-distributive.');

    for object in objectSet:
      if construct_right_tables(
           multCayTab, addCayTab, object) are not equal
      {
        rightDistributive = false;
        print('Not right-distributive.');
        break;
      }
    if (rightDistributive)
      print('Right-distributive');
    if (leftDistributive and rightDistributive)
      print('Distributive.');
  }
}
```

Листинг 7: Псевдокод алгоритма.

Код программы проверки свойства дистрибутивности операции.

```
bool constructLeftTables (vector<vector<int>> mCT,
                          vector<vector<int>> aCT, int curEl)
{
  int j, k;
  vector<vector<int>>
    lTable (matrixDimension, vector<int> (matrixDimension, 0)),
    rTable (matrixDimension, vector<int> (matrixDimension, 0));
  for (j = 0; j < matrixDimension; ++j)
    for (k = 0; k < matrixDimension; ++k)
      {
        lTable[j][k] = mCT[curEl][aCT[j][k]];
        rTable[j][k] = aCT[mCT[curEl][j]][mCT[curEl][k]];
      }
  return (lTable == rTable);
}

bool constructRightTables (vector<vector<int>> mCT,
                           vector<vector<int>> aCT, int curEl)
{
```

```
  int j, k;
  vector<vector<int>>
    lTable (matrixDimension, vector<int> (matrixDimension, 0)),
    rTable (matrixDimension, vector<int> (matrixDimension, 0));
  for (j = 0; j < matrixDimension; ++j)
    for (k = 0; k < matrixDimension; ++k)
      {
        lTable[j][k] = mCT[aCT[j][k]][curEl];
        rTable[j][k] = aCT[mCT[j][curEl]][mCT[k][curEl]];
      }
  return (lTable == rTable);
}

void checkDistributivityMachinerie (
  vector<vector<int>> mCT, vector<vector<int>> aCT)
{
  int i, j, k;
  bool flagL = true, flagR = true;
  if (checkCommutativity (mCT))
    {
      for (i = 0; i < matrixDimension; ++i)
        if (!constructLeftTables (mCT, aCT, i))
          {
            cout << "OPERATION IS NOT DISTRIBUTIVE.\n";
            return;
          }
      cout << "OPERATION IS DISTRIBUTIVE.\n";
    }
  else
    {
      for (i = 0; i < matrixDimension; ++i)
        if (!constructLeftTables (mCT, aCT, i))
          {
            cout << "OPERATION IS NOT LEFT-DISTRIBUTIVE.\n";
            flagL = false;
            break;
          }
      if (flagL)
        cout << "OPERATION IS LEFT-DISTRIBUTIVE.\n";

      for (i = 0; i < matrixDimension; ++i)
```

```
61          if (!constructRightTables (mCT, aCT, i))
62            {
63              cout << "OPERATION IS NOT RIGHT-DISTRIBUTIVE.\n";
64              flagR = false;
65              break;
66            }
67        if (flagR)
68          cout << "OPERATION IS RIGHT-DISTRIBUTIVE.\n";
69        if (flagL && flagR)
70          cout << "OPERATION IS DISTRIBUTIVE.\n";
71      }
72  }
```

Листинг 8: Код программы.

Результат тестирования программы проверки свойства дистрибутивности операции.

Рассмотрим таблицы Кэли операций $+$ и $\cdot$. Проверим дистрибутивность операции $\cdot$ относительно операции $+$.

```
INPUT MATRIX DIMENSION:
4
INPUT THE CAYLEY TABLE OF THE MULTIPLICATION (*) OPERATION:
0 0 0 0
0 1 2 3
0 2 0 2
0 3 2 1
INPUT THE CAYLEY TABLE OF THE ADDITION (+) OPERATION:
0 1 2 3
1 2 3 0
2 3 0 1
3 0 1 2
OPERATION IS DISTRIBUTIVE.
```

Рисунок 2 – Результат проверки свойства дистрибутивности операции $\cdot$ относительно операции $+$.

Как видно, операция $\cdot$ является дистрибутивной относительно операции $+$.

Оценка сложности алгоритма проверки свойства дистрибутивности операции.

Аналогично алгоритму проверки ассоциативности, в этом алгоритме происходит сравнение всех троек аргументов и сравнение элементов правых и левых таблиц дистрибутивности. Временная сложность алгоритма: $O(n^3)$, где n – размер множества объектов, над которым определены операции $\cdot$ и $+$.

3.5 Алгоритм объединения бинарных отношений

Описание алгоритма объединения бинарных отношений.

Вход: матрицы A и B (одинаковой размерности $N \times M$) бинарных отношений ρ и δ.

Выход: матрица $A \cup B$ размерности $N \times M$ бинарного отношения $\rho \cup \delta$.

Метод: вычисляется результат применения операции «логическое ИЛИ» к элементам матриц A и B, расположенным на пересечении i-ой строки и j-го столбца ($i = 1, \ldots, N,\ j = 1, \ldots, M$). Результат применения операции заносится в соответствующую ячейку матрицы $A \cup B$.

Псевдокод алгоритма объединения бинарных отношений.

```
binary_relation_union(<N×M matrix> fBinRelMat,
                      <N×M matrix> sBinRelMat)
{
  <N×M matrix> binRelUnion;
  for i in range(N):
    for j in range(M):
      if (fBinRelMat[i][j] == 1) || (sBinRelMat[i][j] == 1):
        then binRelUnion[i][j] = 1; else binRelUnion[i][j] = 0;
  return (binRelUnion);
}
```

Листинг 9: Псевдокод алгоритма.

Код программы объединения бинарных отношений.

```
vector<vector<int>> getBinaryRelationUnion (
  vector<vector<int>> fM, vector<vector<int>> sM)
{
    int i, j, rowsNum = fM.size (), colsNum = fM[0].size ();
    vector<vector<int>> unionMatrix (
      rowsNum, vector<int> (colsNum, 0));

    for (i = 0; i < rowsNum; ++i)
        for (j = 0; j < colsNum; ++j)
            unionMatrix[i][j] = ((fM[i][j] == 1) ||
                                 (sM[i][j] == 1) ? 1 : 0);
    return (unionMatrix);
}
```

Листинг 10: Код программы.

Результат тестирования программы объединения бинарных отношений.

Рассмотрим пару бинарных отношений ρ и δ, заданных матрицами размерности 4×3. Применим операцию объединения к бинарным отношениям ρ и δ.

```
INPUT THE FIRST DIMENSION OF YOUR BINARY RELATION MATRIX:
4
INPUT THE SECOND DIMENSION OF YOUR BINARY RELATION MATRIX:
3
INPUT THE FIRST BINARY RELATION MATRIX:
1 0 0
0 1 0
0 0 1
1 0 1
INPUT THE SECOND BINARY RELATION MATRIX:
0 1 1
1 0 1
1 1 0
0 1 0
THE RESULT OF UNION OF YOUR BINARY RELATIONS IS:
1 1 1
1 1 1
1 1 1
1 1 1
```

Рисунок 3 – Результат построения объединения бинарных отношений ρ и δ.

В результате работы программы была сформирована матрица бинарного отношения $\rho \cup \delta$.

Оценка сложности алгоритма объединения бинарных отношений.

Для формирования матрицы $A \cup B$ бинарного отношения $\rho \cup \delta$ требуется произвести операции над каждым элементом матриц A и B. Общее число элементов в каждой матрице – $N \times M$, откуда и следует оценка временной сложности алгоритма: $T(get\_binary\_relation\_union) = O(N \cdot M)$.

3.6 Алгоритм пересечения бинарных отношений

Описание алгоритма пересечения бинарных отношений.

Вход: матрицы A и B (одинаковой размерности $N \times M$) бинарных отношений ρ и δ.

Выход: матрица $A \cap B$ размерности $N \times M$ бинарного отношения $\rho \cap \delta$.

Метод: вычисляется результат применения операции «логическое И» к элементам матриц A и B, расположенным на пересечении i-ой строки и j-го столбца ($i = 1, \ldots, N,\ j = 1, \ldots, M$). Результат применения операции заносится в соответствующую ячейку матрицы $A \cap B$.

Псевдокод алгоритма объединения бинарных отношений.

```
binary_relation_intersection(<N×M matrix> fBinRelMat,
                             <N×M matrix> sBinRelMat)
{
  <N×M matrix> binRelIntersection;
  for i in range(N):
    for j in range(M):
      if (fBinRelMat[i][j] == 1) && (sBinRelMat[i][j] == 1):
        then binRelIntersection[i][j] = 1;
        else binRelIntersection[i][j] = 0;
  return (binRelIntersection);
}
```

Листинг 11: Псевдокод алгоритма.

Код программы объединения бинарных отношений.

```
vector<vector<int>> getBinaryRelationIntersection (
  vector<vector<int>> fM, vector<vector<int>> sM)
{
  int i, j, rowsNum = fM.size (), colsNum = fM[0].size ();
  vector<vector<int>> intersectionMatrix (
    rowsNum, vector<int> (colsNum, 0));
  for (i = 0; i < rowsNum; ++i)
    for (j = 0; j < colsNum; ++j)
      intersectionMatrix[i][j] = ((fM[i][j] == 1) &&
                                  (sM[i][j] == 1) ? 1 : 0);
  return (intersectionMatrix);
}
```

Листинг 12: Код программы.

Результат тестирования программы пересечения бинарных отношений.

Рассмотрим пару бинарных отношений ρ и δ, заданных матрицами размерности 3×4. Применим операцию пересечения к бинарным отношениям ρ и δ.

```
INPUT THE FIRST DIMENSION OF YOUR BINARY RELATION MATRIX:
3
INPUT THE SECOND DIMENSION OF YOUR BINARY RELATION MATRIX:
4
INPUT THE FIRST BINARY RELATION MATRIX:
1 0 1 0
0 1 0 1
1 0 1 0
INPUT THE SECOND BINARY RELATION MATRIX:
0 1 0 1
1 0 1 0
0 1 0 1
THE RESULT OF INTERSECTION OF YOUR BINARY RELATIONS IS:
0 0 0 0
0 0 0 0
0 0 0 0
```

Рисунок 4 – Результат построения пересечения бинарных отношений ρ и δ.

В результате работы программы была сформирована матрица бинарного отношения $\rho \cap \delta$.

Оценка сложности алгоритма пересечения бинарных отношений.

Для формирования матрицы $A \cap B$ бинарного отношения $\rho \cap \delta$ требуется произвести операции над каждым элементом матриц A и B. Общее число элементов в каждой матрице – $N \times M$, откуда и следует оценка временной сложности алгоритма: $T(get\_binary\_relation\_intersection) = O(N \cdot M)$.

3.7 Алгоритм построения дополнения бинарного отношения

Описание алгоритма построения дополнения бинарного отношения.

Вход: матрица A размерности $N \times M$ бинарного отношения ρ.

Выход: матрица $\overline{A}$ размерности $N \times M$ бинарного отношения $\overline{\rho}$.

Выход: к каждому элементу матрицы A, стоящему на пересечении i-й строки и j-го столбца ($i = 1, \ldots, N,\ j = 1, \ldots, M$), применяется операция «логическое НЕ» (единица заменяется на ноль и наоборот). Результат применения операции заносится в соответствующую ячейку матрицы $\overline{A}$.

Псевдокод алгоритма построения дополнения бинарного отношения.

```
binary_relation_complement(<N×M matrix> BinRelMat)
{
  <N×M matrix> binRelComplement;
  for i in range(N):
    for j in range(M):
      binRelComplement[i][j] = ¬binRelMat[i][j]
  return (binRelIntersection);
}
```

Листинг 13: Псевдокод алгоритма.

Код программы построения дополнения бинарного отношения.

```
vector<vector<int>> getBinaryRelationComplement (
  vector<vector<int>> binRelMat)
{
  int i, j, rowsNum = binRelMat.size (),
            colsNum = binRelMat[0].size ();
  vector<vector<int>> complementMatrix (
    rowsNum, vector<int> (colsNum, 0));

  for (i = 0; i < rowsNum; ++i)
    for (j = 0; j < colsNum; ++j)
      complementMatrix[i][j] = 1 - binRelMat[i][j];
  return (complementMatrix);
}
```

Листинг 14: Код программы.

Результат тестирования программы построения дополнения бинарного отношения.

Рассмотрим бинарное отношение ρ, заданное матрицей A размерности 3 $\times$ 4. Построим дополнение $\overline{\rho}$ заданного бинарного отношения ρ.

```
INPUT THE FIRST DIMENSION OF YOUR BINARY RELATION MATRIX:
3
INPUT THE SECOND DIMENSION OF YOUR BINARY RELATION MATRIX:
4
INPUT THE BINARY RELATION MATRIX:
1 0 1 0
0 1 0 1
1 0 1 0
THE COMPLEMENT OF YOUR BINARY RELATIONS IS:
0 1 0 1
1 0 1 0
0 1 0 1
```

Рисунок 5 – Результат построения дополнения бинарного отношения ρ.

В результате работы программы было построено бинарное отношение $\overline{\rho}$ с получившейся матрицей $\overline{A}$ размерности 3 $\times$ 4.

Оценка временной сложности алгоритма построения дополнения бинарного отношения.

Поскольку в ходе работы программы обрабатывается каждый элемент некоторой матрицы A размерности $N \times M$, то из этого получаем временную оценку сложности алгоритма: $T(get\_binary\_relation\_complement) = O(N \cdot M)$.

3.8 Алгоритм построения композиции бинарных отношений

Описание алгоритма построения композиции бинарных отношений.

Вход: матрица A размерности $N \times M$ и матрица B размерности $L \times K$ бинарных отношений ρ и δ.

Выход: сообщение об ошибке, в случае если M не равно L, или матрица $A \circ B$ размерности $N \times K$ бинарного отношения $\rho \circ \delta$.

Метод: будем считать, что M и L совпали. Тогда строим матрицу $A \circ B$ путём обычного перемножения матриц A и B: $A \cdot B$, а затем проверяем каждый элемент получившейся матрицы, стоящий на пересечении i-ой строки и j-го столбца ($i = 1, \ldots, N,\ j = 1, \ldots, K$) на то, больше ли он единицы. Если да, то заменяем этот элемент на единицу, если нет, то оставляем без изменения.

Псевдокод алгоритма построения композиции бинарных отношений.

```
1  binary_relation_composition(<N×M matrix> fBinRelMat,
2                              <L×K matrix> sBinRelMat)
3  {
4    if M != L:
5      return <<Wrong Input>>;
6
7    <N × K matrix> boolMultiplicationMatrix, int product;
8    for i in range(N):
9      for j in range(K):
10     {
11       product = 0;
12       for k in range(M):
13         product += fBinRelMat[i][k] * sBinRelMat[k][j];
14       if (product > 0):
15       then boolMultiplicationMatrix[i][j] = 1;
16       else boolMultiplicationMatrix[i][j] = 0;
17     }
18   return (boolMultiplicationMatrix);
19 }
```

Листинг 15: Псевдокод алгоритма.

Код программы построения композиции бинарных отношений.

```
1  vector<vector<int>> getBinaryRelationMultiplication (
2    vector<vector<int>> fBinRel, vector<vector<int>> sBinRel)
3  {
4    int i, j, k, product, rowsNum = fBinRel.size (),
```

```
                        colsNum = sBinRel[0].size ();
  vector<vector<int>> boolMultiplicationMatrix (
    rowsNum, vector<int> (colsNum, 0));

  for (i = 0; i < rowsNum; ++i)
    for (j = 0; j < colsNum; ++j)
      {
        product = 0;
        for (k = 0; k < fBinRel[0].size (); ++k)
          product += fBinRel[i][k] * sBinRel[k][j];
        boolMultiplicationMatrix[i][j] = (product > 0 ? 1 : 0);
      }
  return (boolMultiplicationMatrix);
}
```

Листинг 16: Код программы.

Результат тестирования программы построения композиции бинарных отношений.

Рассмотрим бинарные отношения ρ и δ, заданные матрицами A и B размерности 3×4 и 4×5. Построим композицию этих бинарных отношений.

```
INPUT THE FIRST DIMENSION OF YOUR FIRST BINARY RELATION MATRIX:
3
INPUT THE SECOND DIMENSION OF YOUR FIRST BINARY RELATION MATRIX:
4
INPUT THE FIRST DIMENSION OF YOUR SECOND BINARY RELATION MATRIX:
4
INPUT THE SECOND DIMENSION OF YOUR SECOND BINARY RELATION MATRIX:
5
INPUT THE FIRST BINARY RELATION MATRIX:
0 1 1 1
1 0 1 1
1 1 0 0
INPUT THE SECOND BINARY RELATION MATRIX:
1 1 0 0 0
0 1 1 0 0
0 0 1 1 0
0 0 0 1 1
THE RESULT OF BINARY RELATION MULTIPLICATION IS:
0 1 1 1 1
1 1 1 1 1
1 1 1 0 0
```

Рисунок 6 – Результат построения композиции бинарных отношений.

В ходе выполнения программы была построена матрица $A \circ B$ размерности 3×5, соответствующая бинарному отношению $\rho \circ \delta$.

Оценка временной сложности алгоритма построения композиции бинарных отношений.

Рассмотрим матрицы A размерности $N \times M$ и B размерности $M \times K$ бинарных отношений ρ и δ. В матрице $A \circ B$ композиции бинарных отношений будет содержаться $N \times K$ элементов. Для вычисления значения каждого элемента матрицы $A \circ B$, находящегося на пересечении i-ой строки и j-го столбца ($i = 1, \ldots, N,\ j = 1, \ldots, K$), необходимо поэлементно перемножить i-ую строку первой матрицы с j-ым столбцом второй матрицы. Для нахождения значения такого произведения требуется осуществить M операций. Из этого следует временная оценка сложности: $T(get\_binary\_relation\_multiplication) = O(N \cdot M \cdot K)$.

3.9 Алгоритм обращения бинарного отношения

Описание алгоритма обращения бинарного отношения.

Вход: матрица A размерности $N \times M$ бинарного отношения ρ.

Выход: матрица A^T размерности $M \times N$ бинарного отношения ρ^{-1}.

Метод: необходимо осуществить транспонирование исходной матрицы A бинарного отношения ρ.

Псевдокод алгоритма обращения бинарного отношения.

```
binary_relation_inversion(<N×M matrix> binRelMat)
{
  return (transpose(binRelMat));
}
```

Листинг 17: Псевдокод алгоритма.

Код программы обращения бинарного отношения.

```
vector<vector<int>> transposeBinaryRelationMatrix (
  vector<vector<int>> binRelMat)
{
  int i, j;
  vector<vector<int>> transposedMatrix;

  for (i = 0; i < binRelMat[0].size (); ++i)
    {
      vector<int> column;
      for (j = 0; j < binRelMat.size (); ++j)
        column.push_back (binRelMat[j][i]);
      transposedMatrix.push_back (column);
    }
  return (transposedMatrix);
}
```

Листинг 18: Псевдокод алгоритма.

Результат тестирования программы обращения бинарного отношения.

Рассмотрим бинарное отношение ρ, заданное матрицей A размерности 3 $\times$ 4. Найдём обратное к этому отношение.

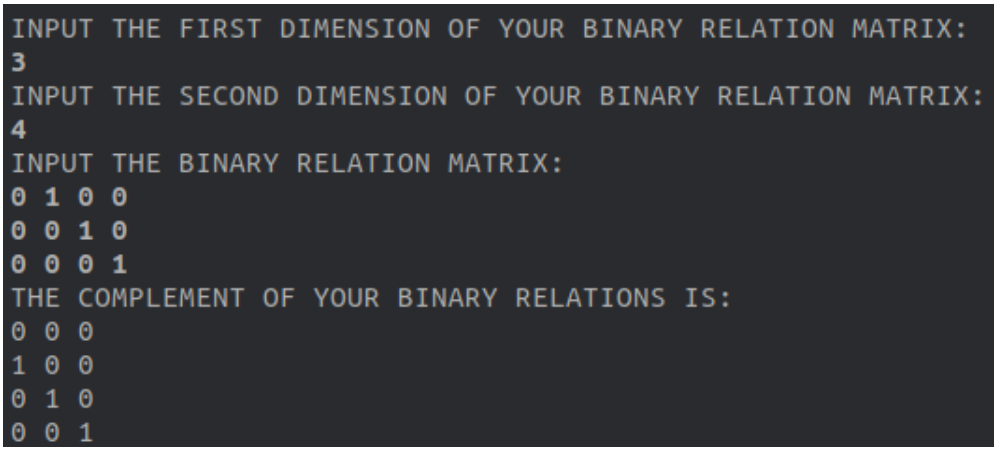


Рисунок 7 – Результат обращения бинарного отношения.

В результате работы программы была построена матрица A^T размерности 4 $\times$ 3 бинарного отношения ρ^{-1}.

Оценка временной сложности алгоритма обращения бинарного отношения.

Оценим временную сложность алгоритма на примере матрицы A размерности $N \times M$ бинарного отношения ρ. После обращения бинарного отношения ρ и построения бинарного отношения ρ^{-1} число элементов в матрице A^T будет соответствовать числу элементов в матрице A. Поскольку меняется только размерность матрицы, а сами элементы лишь меняют своё расположение, то получаем оценку временной сложности алгоритма: $T(get\_binary\_relation\_inversion) = O(N \cdot M)$.

3.10 Алгоритм сложения матриц в конечном поле

Описание алгоритма сложения двух матриц в конечном поле.

Вход: матрицы A и B размерности $N \times M$ над множеством натуральных чисел и натуральное число ord – порядок поля.

Выход: матрица $A + B$ размерности $N \times M$ в конечном поле порядка ord.

Метод: сложение матриц A и B осуществляется по обычному закону сложения матриц. Затем в каждую ячейку, стоящую на пересечении i-ой строки и j-го столбца ($i = 1, \ldots, N,\ j = 1, \ldots, M$) получившейся матрицы, записывается число – результат взятия остатка от деления рассматриваемого элемента на порядок поля.

Псевдокод алгоритма сложения матриц в конечном поле.

```
matrix_field_addition(<N×M matrix> fMat,
                      <N×M matrix> sMat,
                      unsigned int fieldOrder)
{
  <N×M matrix> resMat;
  for i in range(N):
    for j in range(M):
      resMat[i][j] = (fMat[i][j] + sMat[i][j]) % fieldOrder;
  return (resMat);
}
```

Листинг 19: Псевдокод алгоритма.

Код программы сложения матриц в конечном поле.

```
vector<vector<int>> addMatricesMachinerie (
  vector<vector<int>> lM,
  vector<vector<int>> rM, int fieldOrder)
{
  int i, j;
  vector<vector<int>> resM (
    lM.size (), vector<int> (lM[0].size (), 0));

  for (i = 0; i < lM.size (); ++i)
    for (j = 0; j < lM[i].size (); ++j)
      resM[i][j] = (lM[i][j] + rM[i][j]) % fieldOrder;
  return (resM);
}
```

Листинг 20: Код программы.

Результат тестирования программы сложения матриц в конечном поле.

Рассмотрим матрицы A и B размерности 3×4 и порядок поля ord, равный 8.

```
INPUT MATRIX FIRST DIMENSION:
3
INPUT MATRIX SECOND DIMENSION:
4
INPUT THE ORDER OF A FINITE FIELD:
8
INPUT THE ELEMENTS OF THE FIRST MATRIX:
1 2 3 4
5 6 7 8
9 0 1 2
INPUT THE ELEMENTS OF THE SECOND MATRIX:
5 4 3 6
2 1 0 7
9 8 7 8
THE RESULT OF MATRIX ADDITION IS:
6 6 6 2
7 7 7 7
2 0 0 2
```

Рисунок 8 – Результат сложения матриц в конечном поле порядка 8

.

Результатом работы программы является матрица $A + B$ в конечном поле порядка 8.

Оценка временной сложности алгоритма сложения двух матриц в конечном поле.

Для формирования матрицы $A + B$ в конечном поле порядка ord требуется произвести операции над каждым элементом матриц A и B. Общее число элементов в каждой матрице – $N \times M$, откуда и следует оценка временной сложности алгоритма: $T(get\_matrix\_field\_addition) = O(N \cdot M)$.

3.11 Алгоритм умножения матриц в конечном поле

Описание алгоритма умножения матриц в конечном поле.

Вход: матрицы A размерности $N \times M$ и B размерности $L \times K$ над множеством натуральных чисел и натуральное число – порядок поля ord.

Выход: сообщение об ошибке в случае, если размерность M не совпадает с L или матрица $A \cdot B$ над конечным полем порядка ord.

Метод: в случае совпадения размерностей M и L умножение матриц A и B осуществляется по обычному закону умножения матриц. Затем в каждую ячейку, стоящую на пересечении i-ой строки и j-го столбца ($i = 1, \ldots, N,\ j = 1, \ldots, K$) получившейся матрицы, записывается число – результат взятия остатка от деления рассматриваемого элемента на порядок поля.

Псевдокод алгоритма умножения матриц в конечном поле.

```
matrix_field_multiplication(<N×M matrix> fMat,
                            <L×K matrix> sMat)
{
  if M != L:
    return <<Wrong Input>>;

  <N × K matrix> multiplicationMat, int fieldOrder;
  for i in range(N):
    for j in range(K):
    {
      product = 0;
      for k in range(M):
        product += fMat[i][k] * sMat[k][j];
      multiplicationMat[i][j] = (product % fieldOrder);
    }
  return (multiplicationMat);
}
```

Листинг 21: Псевдокод алгоритма.

Код программы умножения матриц в конечном поле.

```
vector<vector<int>> multiplyMatricesMachinerie (
  vector<vector<int>> lM, vector<vector<int>> rM, int fieldOrder)
{
  int i, j, k;
  vector<vector<int>> resM (lM.size (),
                            vector<int> (rM[0].size (), 0));
```

```
7
8   for (i = 0; i < lM.size (); ++i)
9     for (j = 0; j < rM[0].size (); ++j)
10      {
11        int product = 0;
12        for (k = 0; k < lM[0].size (); ++k)
13          product += (lM[i][k] * rM[k][j]);
14        resM[i][j] = (product % fieldOrder);
15      }
16  return (resM);
17 }
```

Листинг 22: Код программы.

Результат тестирования программы умножения матриц в конечном поле.

Рассмотрим матрицы A размерности 3×4 и B размерности 4×3 и порядок поля ord, равный 13.

```
INPUT THE FIRST DIMENSION OF THE FIRST MATRIX:
3
INPUT THE SECOND DIMENSION OF THE FIRST MATRIX:
4
INPUT THE FIRST DIMENSION OF THE SECOND MATRIX:
4
INPUT THE SECOND DIMENSION OF THE SECOND MATRIX:
3
INPUT THE ORDER OF A FINITE FIELD:
13
INPUT THE ELEMENTS OF THE FIRST MATRIX:
0 1 2 3
4 5 6 7
8 9 5 6
INPUT THE ELEMENTS OF THE SECOND MATRIX:
7 5 3 2
9 7 6 8
5 6 9 2
THE RESULT OF MATRIX MULTIPLICATION IS:
6 0 10
12 7 0
10 7 7
```

Рисунок 9 – Результат умножения двух матриц в конечном поле порядка 13.

Результатом работы программы является матрица $A \cdot B$ размерности 3×3 в конечном поле порядка 13.

Оценка временной сложности алгоритма умножения матриц в конечном поле.

Рассмотрим матрицы A размерности $N \times M$ и B размерности $M \times K$ и порядок поля ord. В матрице $A \cdot B$ в конечном поле порядка ord будет содержаться $N \times K$ элементов. Для вычисления значения каждого элемента матрицы $A \cdot B$, находящегося на пересечении i-ой строки и j-го столбца ($i = 1, \ldots, N,\ j = 1, \ldots, K$), необходимо поэлементно перемножить i-ую строку первой матрицы с j-ым столбцом второй матрицы, после чего взять остаток от деления полученного значения на порядок поля. Для нахождения значения произведения требуется осуществить M операций. Из этого следует временная оценка сложности: $T(get\_matrix\_field\_multiplication) = O(N \cdot M \cdot K)$.

3.12 Алгоритм транспонирования матрицы в конечном поле

Описание алгоритма транспонирования матрицы в конечном поле.

Вход: матрица A над множеством натуральных чисел размерности $N \times M$ и натуральное число – порядок поля ord.

Выход: матрица A^T размерности $N \times M$ в конечном поле порядка ord.

Метод: каждый столбец исходной матрицы A становится строкой, а строка – наоборот, столбцом матрицы A^T. Значение каждой ячейки матрицы A^T заменяется на результат взятия остатка от деления соответствующего элемента на порядок поля.

Псевдокод алгоритма транспонирования матрицы в конечном поле.

```
matrix_field_transposition(<N × M matrix> mat)
{
  <M × N matrix> transposedMat;
  for each column in mat:
    transposedMat.push_back(make_row(
                              reduce_column(column, fieldOrder)))
  return transposedMat;
}
```

Листинг 23: Псевдокод алгоритма.

Код программы транспонирования матрицы в конечном поле.

```
vector<vector<int>> transposeMatrixMachinerie (
  vector<vector<int>> matrix, int fieldOrder)
{
  int i, j;
  vector<vector<int>> tM (matrix[0].size (),
                          vector<int> (matrix.size (), 0));

  for (i = 0; i < matrix[0].size (); ++i)
    {
      vector<int> column;
      for (j = 0; j < matrix.size (); ++j)
        column.push_back (matrix[j][i] % fieldOrder);
      tM[i] = column;
    }
  return (tM);
}
```

Листинг 24: Код программы.

Результат тестирования программы транспонирования матрицы в конечном поле.

Рассмотрим матрицу A над множеством натуральных чисел размерности 5×3 и порядок поля ord, равный 7.

```
INPUT THE FIRST DIMENSION OF THE MATRIX:
5
INPUT THE SECOND DIMENSION OF THE MATRIX:
3
INPUT THE ORDER OF A FINITE FIELD:
7
INPUT THE ELEMENTS OF THE MATRIX:
0 1 2
3 4 5
6 7 8
9 8 7
6 5 4
THE RESULT OF MATRIX TRANSPOSITION IS:
0 3 6 2 6
1 4 0 1 5
2 5 1 0 4
```

Рисунок 10 – Результат транспонирования матрицы в конечном поля порядка 7.

Результатом транспонирования матрицы A размерности 5×3 является матрица A^T над конечным полем порядка 7 размерности 3×5.

Оценка временной сложности алгоритма транспонирования матрицы в конечном поле.

Оценим временную сложность алгоритма на примере матрицы A размерности $N \times M$ и порядка поля ord. После транспонирования матрицы A и построения матрицы A^T над конечным полем порядка ord число элементов в матрице A^T будет соответствовать числу элементов в матрице A. Поскольку меняется только размерность матрицы, а сами элементы лишь меняют своё расположение, то получаем оценку временной сложности алгоритма: $T(get\_matrix\_field\_transposition) = O(N \cdot M)$.

3.13 Алгоритм обращения матрицы над конечным полем

Описание алгоритма обращения матрицы над конечным полем.

Вход: квадратная матрица A размерности $N \times N$ над множеством натуральных чисел и натуральное число – порядок поля ord.

Выход: квадратная матрица A^{-1} размерности $N \times N$ над конечным полем порядка ord – обратная к A матрица или сообщение о том, что матрица A является вырожденной.

Метод: сначала вычисляется определитель $\det A$ исходной матрицы A. Если он кратен порядку поля, то матрица A является вырожденной, и выполнение программы прекращается. Иначе формируется сопряжённая к A матрица A^* – матрица алгебраических дополнений. По расширенному алгоритму Евклида для $\det A$ в поле заданного порядка ищется обратный элемент, который умножается на A^*. Если какой-то элемент в получившейся матрице отрицательный, то обновляем его значение по следующей формуле: $matInverse[i][j] = (matInverse \ \% \ fieldOrder) + fieldOrder$.

Псевдокод алгоритма обращения матрицы над конечным полем.

```
matrix_field_inversion(<N×M matrix> mat, unsigned int fieldOrder)
{
  det = (compute_det(mat) % fieldOrder) + fieldOrder;
  if (det % fieldOrder == 0):
    return;
  detFieldInverse = get_field_inverse(det, fieldOrder);
  <N×N matrix> conjugateMatrix = get_conjugate_matrix(mat);
  return (remove_negatives(detFieldInverse * conjugateMatrix,
                           fieldOrder));
}
```

Листинг 25: Псевдокод алгоритма.

В программной реализации алгоритма в качестве вспомогательных используются следующие функции: $getMinor$, извлекающая необходимый минор при вычислении определителя, $gcdExtended$, реализующая расширенный алгоритм Евклида нахождения обратного к вычисленному определителю элемента в конечном поле заданного порядка, $getMinorExtended$, извлекающая необходимый минор при построении матрицы алгебраических дополнений

Код программы обращения матрицы над конечным полем.

```
vector<vector<int>> getMinor (int columnIdx,
                              vector<vector<int>> matrix)
{
  int i, j;
  vector<vector<int>> minor;
  for (i = 1; i < matrix.size(); ++i)
    {
      vector<int> row;
      for (j = 0; j < matrix.size(); ++j)
        if (j != columnIdx)
          row.push_back (matrix[i][j]);
      minor.push_back (row);
    }
  return (minor);
}

int computeDet (vector<vector<int>> matrix)
{
  if (matrix.size() == 1)
    return (matrix[0][0]);
  int det = 0, multiplier = 1, i;
  for (i = 0; i < matrix.size(); ++i)
    {
      int elt = matrix[0][i];
      if (elt != 0)
        det += multiplier * elt *
                            computeDet (getMinor (i, matrix));
      multiplier *= -1;
    }
  return (det);
}

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;
}

vector<vector<int>> getMinorExtended (vector<vector<int>> matrix,
                                      int rowIdx, int colIdx)
{
  int i, j, matrixSize = matrix.size ();
  vector<vector<int>> minor;

  for (i = 0; i < matrixSize; ++i)
    {
      if (i == rowIdx) continue;
      vector<int> row;
      for (j = 0; j < matrixSize; ++j)
        {
          if (j == colIdx) continue;
          row.push_back (matrix[i][j]);
        }
      minor.push_back (row);
    }
  return (minor);
}

vector<vector<int>> getConjugateMatrix (
  vector<vector<int>> matrix)
{
  int i, j;
  short sign = 1;
  bool evenDimensionFlag = (matrix.size () % 2 == 0);
  vector<vector<int>> conjugateMatrix (
    matrixDimension, vector<int> (matrixDimension, 0));

  for (i = 0; i < matrixDimension; ++i)
    {
      for (j = 0; j < matrixDimension; ++j)
        {
          conjugateMatrix[i][j] =
            sign * computeDet (getMinorExtended (matrix, i, j));
          sign *= -1;
        }
      sign *= (evenDimensionFlag ? -1 : 1);
```

```
82      }
83    return (transposeMatrixMachinerie (conjugateMatrix));
84  }
85
86  void reduceMatrix (
87    vector<vector<int>>& matrix, int fieldOrder, int detInverse)
88  {
89    int i, j;
90
91    for (i = 0; i < matrix.size (); ++i)
92      for (j = 0; j < matrix.size (); ++j)
93        {
94          matrix[i][j] *= detInverse;
95          matrix[i][j] %= fieldOrder;
96          if (matrix[i][j] < 0)
97            matrix[i][j] += fieldOrder;
98        }
99  }
100
101 void invertMatrix ()
102 {
103   unsigned int fieldOrder;
104   int det, detInverse = 0, dummyVariable = 0;
105   vector<vector<int>> inputMatrix, conjugateMatrix;
106
107   cout << "INPUT THE DIMENSION OF YOUR MATRIX:\n";
108   cin >> matrixDimension;
109   cout << "INPUT THE FIELD ORDER:\n";
110   cin >> fieldOrder;
111   cout << "INPUT YOUR MATRIX:\n";
112   getMatrix (inputMatrix);
113
114   det = computeDet (inputMatrix);
115   if (det % fieldOrder == 0)
116     {
117       cout << "MATRIX HAS ZERO DETERMINANT.
118         INVERSE MATRIX CANNOT BE FOUND.\n";
119       return;
120     }
121   det = (det % fieldOrder) + fieldOrder;
122   gcdExtended (det, fieldOrder, detInverse, dummyVariable);
```

```
123   conjugateMatrix = getConjugateMatrix (inputMatrix);
124   reduceMatrix (conjugateMatrix, fieldOrder, detInverse);
125   cout << "INVERSE MATRIX IS:\n";
126   displayMatrix (conjugateMatrix);
127 }
```

Листинг 26: Код программы.

Результат тестирования программы обращения матрицы над конечным полем.

Рассмотрим невырожденную квадратную матрицу A размерности $N \times N$ в конечном поле порядка 37. Найдём обратную к A матрицу.

```
INPUT THE DIMENSION OF YOUR MATRIX:
3
INPUT THE FIELD ORDER:
37
INPUT YOUR MATRIX:
11 15 12
15 2 15
17 15 19
INVERSE MATRIX IS:
5 22 34
1 6 18
22 34 8
```

Рисунок 11 – Результат обращения матрицы в конечном поле порядка 37.

Проверим, что полученная матрица действительно является обратной к A. При умножении исходной матрицы и полученной должна получиться единичная матрица.

```
INPUT THE FIRST DIMENSION OF THE FIRST MATRIX:
3
INPUT THE SECOND DIMENSION OF THE FIRST MATRIX:
3
INPUT THE FIRST DIMENSION OF THE SECOND MATRIX:
3
INPUT THE SECOND DIMENSION OF THE SECOND MATRIX:
3
INPUT THE ORDER OF A FINITE FIELD:
37
INPUT THE ELEMENTS OF THE FIRST MATRIX:
11 15 12
15 2 15
17 15 19
INPUT THE ELEMENTS OF THE SECOND MATRIX:
5 22 34
1 6 18
22 34 8
THE RESULT OF MATRIX MULTIPLICATION IS:
1 0 0
0 1 0
0 0 1
```

Рисунок 12 – Проверка корректности нахождения обратной матрицы.

Оценка временной сложности алгоритма обращения матрицы в конечном поле.

Будем рассматривать невырожденную квадратную матрицу A размерности $N \times N$ с порядком поля ord.

Будем оценивать сложность алгоритма в соответствии с операциями, упомянутыми в псевдокоде.

Первая операция – нахождение определителя матрицы A. Для нахождения определителя матрицы порядка $N \times N$ методом Лапласа (разложением по строке или столбцу) необходимо вычислить значения N миноров $N-1$ порядка. Вычисление будет продолжаться, пока миноры не станут 1-го порядка (одноэлементные матрицы). Следовательно, временная сложность операции вычисления определителя матрицы порядка $N \times N$ составляет: $T(compute\_det, N) = O(N!)$.

Вторая операция – нахождение обратного к определителю элемента в конечном поле заданного порядка. Обратный элемент ищется по расширенному алгоритму Евклида. Временная сложность выполнения этой операции составляет: $T(gcdExtended) = O(\log b)$, где b – наименьшее из двух входящих значений (оценка сложности получена в книге: Knuth, Donald. The Art of Computer Programming. Addison-Wesley. Volume 2, Chapter 4).

Третья операция – построение сопряжённой матрицы. Аналогично вычислению определителя матрицы, только теперь дополнительный минор должен быть вычислен для каждого элемента исходной матрицы. Таким образом, временная сложность операции построения сопряжённой матрицы составляет: $T(get\_conjugate\_matrix) = O((N!)^2)$, где N – размерность исходной квадратной матрицы.

Заключительная операция – проход по получившейся матрице с целью замены отрицательных элементов на элементы поля заданного порядка. Временная сложность: $O(N^2)$.

Таким образом, сложность алгоритма обращения невырожденной квадратной матрицы размерности $N \times N$ в конечном поле: $T(get\_inverse\_field\_matrix) = O((N!)^2)$.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы мною были изучены основные понятия универсальной алгебры и операции над бинарными отношениями. В качестве практического задания мною были описаны и реализованы алгоритмы проверки основных свойств операций по их таблице Кэли, алгоритмы построения объединения, пересечения, дополнения, композиции и обращения бинарных отношений, а также алгоритмы сложения, умножения, транспонирования и обращения матриц в полях конечного порядка. Для этих алгоритмов были представлены псевдокоды, осуществлена их программная реализация, а также проведена оценка временной сложности.