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

Отношение эквивалентности и отношение порядка

ОТЧЁТ
ПО ДИСЦИПЛИНЕ
«ПРИКЛАДНАЯ УНИВЕРСАЛЬНАЯ АЛГЕБРА»

студента 3 курса 331 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Яхина Шамиля Илдусовича

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

Саратов 2022

СОДЕРЖАНИЕ

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

1 Теория

1.1 Отношение эквивалентности

Подмножества декартова произведения $A \times B$ множеств A и B называются бинарными отношениями между элементами множеств A, B и обозначаются строчными греческими буквами: $\rho, \sigma, ..., \rho_1, \rho_2, ...$.

Бинарное отношение ε на множестве A называется отношением эквивалентности (или просто *эквивалентностью*), если оно рефлексивно, симметрично и транзитивно.

Для бинарного отношения $\rho \subset A \times B$ область определения D_ρ и множество значений E_ρ определяются как подмножества соответствующих множеств и по следующим формулам:

$D_\rho = \{a : (a, b) \in \rho$ для некоторого $b \in B\}$,

$E_\rho = \{b : (a, b) \in \rho$ для некоторого $a \in A\}$.

Для любого подмножества $X \subset A$ множество $\rho(X) = \{b \in B : (x, b) \in \rho$ для некоторого $x \in X\}$ называется *образом* множества X относительно отношения ρ.

Образ одноэлементного множества $X = \{a\}$ относительно отношения ρ обозначается символом $\rho(a)$ и называется также образом элемента a или *срезом* отношения ρ через элемент a.

Срезы $\varepsilon(a)$ называются *классами эквивалентности* по отношению ε и сокращенно обозначаются символом $[a]$.

Множество всех таких классов эквивалентности $\{[a] : a \in A\}$ называется *фактор-множеством* множества A по эквивалентности ε и обозначается символом A/ε.

1.2 Отношение порядка

Бинарное отношение ω на множестве A называется отношением порядка (или просто *порядком*), если оно рефлексивно, антисимметрично и транзитивно.

Множество A с заданным на нем отношением порядка $\leq$ называется *упорядоченным множеством* и обозначается $A = (A, \leq)$ или просто $(A, \leq)$.

Элемент a упорядоченного множества $(A, \leq)$ называется:

— *минимальным*, если $(\forall x \in A) x \leq a \Rightarrow x = a$,

— *максимальным*, если $(\forall x \in A) a \leq x \Rightarrow x = a$,

— *наименьшим*, если $(\forall x \in A) a \leq x$,

— *наибольшим*, если $(\forall x \in A) x \leq a$.

Упорядоченное множество $A = (A, \leq)$ наглядно представляется *диаграммой Хассе*, которая представляет элементы множества A точками плоскости и пары $a < \cdot b$ представляет линиями, идущими *вверх* от элемента a к элементу b.

Алгоритм построения диаграммы Хассе конечного упорядоченного множества $A = (A, \leq)$.

1. В упорядоченном множестве $A = (A, \leq)$ найти множество A_1 всех минимальных элементов и расположить их в один горизонтальный ряд (это первый уровень диаграммы).
2. В упорядоченном множестве $A \setminus A_1$, найти множество A_2 всех минимальных элементов и расположить их в один горизонтальный ряд над первым уровнем (это второй уровень диаграммы). Соединить отрезками элементы этого ряда с покрываемыми ими элементами предыдущего ряда.
3. В упорядоченном множестве $A \setminus (A_1 \cup A_2)$ найти множество A_3 всех минимальных элементов и расположить их в один горизонтальный ряд над вторым уровнем (это третий уровень диаграммы). Соединить отрезками элементы этого ряда с покрываемыми ими элементами предыдущих рядов.
4. Процесс продолжается до тех пор, пока не выберутся все элементы множества A.

1.3 Определения контекста и концепта

Контекстом называется алгебраическая система $K = (G, M, \rho)$, состоящая из множества *объектов* G, множества *атрибутов* M и бинарного отношения $\rho \subset G \times M$, показывающего $(g, m) \in \rho$, что объект g имеет атрибут m.

Упорядоченная пара (X, Y) замкнутых множеств $X \in Z_{f_G}, Y \in Z_{f_M}$, удовлетворяющих условиям $\varphi(X) = Y$, $\psi(Y) = X$, называется *концептом* контекста $K = (G, M, \rho)$. При этом компонента X называется *объемом* и компонента Y - *содержанием* концепта (X, Y).

Множество всех концептов $C(K)$ так упорядочивается отношением $(X, Y) \leq (X_1, Y_1) \Leftrightarrow X \subset X_1$ (или равносильно $Y_1 \subset Y$), что $(C(K), \leq)$ является полной решеткой, которая изоморфна решетке замкнутых подмножеств множества G.

Алгоритм вычисления системы замыканий на множестве G:

1. Рассматриваем множество $G \in Z_{f_G}$.

2. Последовательно перебираем все элементы $m \in M$ и вычисляем для них $\psi(\{m\}) = \rho^{-1}(m)$.
3. Вычисляем все новые пересечения множества $\psi(\{m\})$ с ранее полученными множествами и добавляем новые множества к Z_{f_G}. Аналогично вычисляется система замыканий на множестве M.

2 Результаты работы

2.1 Алгоритм построения замыкания относительно эквивалентности

Вход: Размерность матрицы N и матрица представления бинарного отношения размерности $N \times N$

Выход: Матрица построенного замыкания относительно эквивалентности размерности $N \times N$.

1. Построение замыкания относительно свойства рефлексивности.
2. Построение замыкания относительно свойства симметричности.
3. Построение замыкания относительно свойства транзитивности.

Временная сложность алгоритма определения построения замыкания эквивалентности = $O(n^4 + n^2 + n)$ = $O(n^4)$

2.2 Алгоритм построения фактор-множества

Вход: Размерность матрицы N и матрица представления бинарного отношения размерности $N \times N$

Выход: Фактор-множество fm бинарного отношения.

1. Запускается цикл for с i от 1 до N - 1.
   а) Создается пустой список vec и добавляется в список списков fm
   б) Запускается цикл for с j от 1 до N - 1. Если a[i][j] = 1, то в последний список из fm добавляется величина j + 1.
2. fm сортируется, убираются одинаковые элементы из fm.
3. Запускается цикл for с i от 1 до размера fm - 1.
   а) Запускается цикл for с j от 1 до размера fm - 1 и в нем выводится fm[i][j].

Временная сложность алгоритма построения фактор-множества = $O(n^2)$

2.3 Алгоритм получения системы представителей фактор-множества

Вход: Фактор-множество f.

Выход: Система представителей фактор-множествая fm бинарного отношения.

1. Запускается цикл for с i от 1 до размера фактор-множества fm - 1.
   а) Выводится fm[i][0].

Временная сложность алгоритма получения системы представителей фактор-множества = $O(n)$

2.4 Построение диаграммы Хассе для отношения порядка, заданного с помощью числа

2.4.1 Алгоритм поиска минимальных элементов и наименьшего элемента

Вход: Число x.

Выход: Минимальные элементы, наименьший элемент.

1. Создаются списки diagramm_el и a_has.
2. Запускается цикл for с i от 2 до заданного числа x.
   а) Если x $\%$ i = 0, то добавляем i в вектор a_has.
3. Если 1 включается в делители, то минимальным и наименьшим элементом является число 1.
4. Если 1 не включается в делители, то создается список mins и запускается цикл for с i от 0 до размера списка делителей a_has - 1.
   а) bool prov = 0.
   б) Запускается цикл for с j от 0 до i - 1.
      i. Если a_has[i] % a_has[j] = 0, то prov = 1
   в) Если prov = 0, то a_has[i] записывается в mins.
5. Выводится список минимальных элементов mins. Наименьший элемент - это первый элемент списка mins.

Временная сложность алгоритма поиска минимальных элементов и наименьшего элемента = $O(n^2)$

2.4.2 Алгоритм построения диаграммы Хассе

Вход: Список списков levels_const с минимальными элементами, распределенными по уровням, список min (первый уровень) и список всех элементов a_has

Выход: Диаграмма Хассе.

1. Пока a_has не пустой:
   а) Если в списке списков минимальных элементов levels_const больше одного списка, то запускается цикл for с i от 0 до размера последнего списка.
      i. Перебирается предпоследний список и, если mins[i] делится на элемент этого списка, то данная пара кладется в список пар relations.
   б) Ищется список минимальных элементов mins.

в) mins кладется в список списков levels_const.

г) Из a_has удаляются все элементы списка mins.

д) Если a_has не пустой, то находятся минимальные элементы для a_has по алгоритму 3.4.1.

2. Выводится список списков levels_const и список пар relations.

Временная сложность алгоритма построения диаграммы Хассе = $O(n^3)$

2.5 Построение диаграммы Хассе для отношения порядка, заданного с помощью матрицы

2.5.1 Алгоритм поиска минимальных элементов и наименьшего элемента

Вход: Размерность матрицы N и матрица представления a размерности $N \times N$

Выход: Минимальные элементы, наименьший элемент.

1. Создается копия матрицы и помещается в a_new.
2. bool prov = 1.
3. Запускается цикл for с k от 0 до N - 1.

   а) min_el = 2 * N;

   б) Очищается dop_l;

   в) Запускается цикл for с i от 0 до N - 1.

   i. prov = true.

   ii. int sk = 0.

   iii. Запускается цикл for с j от 0 до N - 1.

   А. Если a_new[j][i] = 2, то prov = 0 и break;

   Б. Если a_new[j][i] = 1, то sk++;

   iv. Если prov = 1, то в случае, если sk = min_el, в dop_l добавляется i + 1, а если sk < min_el, то min_el = sk, dop_l очищается и в него кладется i + 1.

   г) k уменьшается на 1;

   д) Запускается цикл for с i от 0 до размера dop_l - 1.

   i. Запускается цикл for с j от 0 до N - 1, в котором выполняется a_new[j][dop_l[i] - 1] = 2.

   ii. k увеличивается на 1;

   е) В m_p кладется dop_l;

4. В min_res кладется dop_l;

5. Выводится список минимальных элементов min_res.
6. Если размер min_res больше единицы, то наименьшего элемента нет, а если размер равен единице, то наименьший элемент - min_res[0].

Временная сложность алгоритма поиска минимальных элементов и наименьшего элемента = $O(n^3)$

2.5.2 Алгоритм поиска максимальных элементов и наибольшего элемента

Вход: Размерность матрицы N и матрица представления a размерности $N \times N$, список элементов по уровням m_p

Выход: Максимальные элементы, наибольший элемент.

1. Запускается цикл for с i от 0 до размера m_p - 2.
   а) Запускается цикл for с j от 0 до размера m_p[i] - 1.
      i. bool prov = true.
      ii. Запускается цикл for с k от 0 до размера m_p[i + 1] - 1 и если в нем a[m_p[i][j] - 1][m_p[i + 1][k] - 1] = 1, то prov = false и завершаем цикл
   б) если prov = true, то кладем в max_res элемент m_p[i][j].
2. Все элементы последнего списка из m_p кладутся в список max_res.
3. Выводится список максимальных элементов max_res.
4. Если размер max_res больше единицы, то наибольшего элемента нет, а если размер равен единице, то наибольший элемент - max_res[0].

Временная сложность алгоритма поиска максимальных элементов и наибольшего элемента = $O(n^3)$

2.5.3 Алгоритм построения диаграммы Хассе

Вход: Размерность матрицы N и матрица представления a размерности $N \times N$

Выход: Диаграмма Хассе.

1. По алгоритму 2.5.1. находятся все элементы, распределенные по уровням и помещаются в список списков m_p.
2. Последовательно выводятся все уровни из m_p.
3. Если между двумя соседними уровнями есть такие элементы, что в матрице a на пересечении их индексов стоит 1, они выводятся как элементы, между которыми существует связь.

Временная сложность алгоритма построения диаграммы Хассе = $O(n^3)$

2.6 Алгоритм вычисления решетки концептов

2.6.1 Построение системы замыканий

Вход: Множество объектов G_context, множество атрибутов M_context, размерность матрицы N и матрица представления matr размерности $N \times N$.

Выход: Система замыканий.

1. Создается список списков Z_fG и в него кладется.
2. Создаются списки rho_helper и intersection.
3. Запускается цикл for с i от 0 до N - 1
   а) Очищается список rho_helper.
   б) Запускается цикл for с j от 0 до N - 1 и если в нем matr[j][i] = 1, то j + 1 кладется в rho_helper.
   в) rho_helper кладется в Z_fG.
   г) Запускается цикл for с k от 0 до размера Z_fG[j] - 1 и в нем еще один цикл for с l от 0 до размера rho_helper - 1 и если в нем rho_helper[l] = Z_fG[j][k], то rho_helper[l] кладется в intersection.
   д) intersection кладется в Z_fG.
4. если prov = true, то кладем в max_res элемент m_p[i][j].
5. Сортируется Z_fG[j][k] и удаляются одинаковые элементы
6. Выводится система замыканий Z_fG

Временная сложность алгоритма построения системы замыканий = $O(n^4)$

2.6.2 Построение решетки концептов

Вход: Множество объектов G_context, множество атрибутов M_context, размерность матрицы N, матрица представления matr размерности $N \times N$ и система замыканий Z_fG.

Выход: Решетка концептов.

1. Создается копия системы замыканий Z_fG_copy.
2. Запускается цикл for с y от 0 до размера Z_fG_copy - 1 и если в цикле Z_fG_copy[y].size() = 0.
   а) В reshetka_elems[0] кладется dop_vec
   б) Из Z_fG_copy удаляется элемент с индексом y.
   в) Запускается цикл while, пока Z_fG_copy не станет пустым
      i. Рассматриваются все элементы в Z_fG_copy (i - индекс рассматриваемого вектора, j - индексы всех остальных векторов, k - эле-

менты рассматриваемого вектора, l - элементы остальных векторов)

A. Если Z_fG_copy[i][k] = Z_fG_copy[j][l], то в in_vect добавляется Z_fG_copy[i][k].

Б. Если размер in_vect = размеру Z_fG_copy[i], сравниваются элементы остальных векторов Z_fG_copy[j][h] и элементы в in_vect.

В. Если количество равных элементов = размеру Z_fG_copy[j], то останавливаем цикл, иначе кладем Z_fG_copy[i] в resh_min_elems и кладем в i в список удаляемых элементов delete_el.

ii. Удаляем из Z_fG_copy элементы с индексами, равными элементам в delete_el.

iii. Кладем resh_min_elems в reshetka_elems.

г) Строим решетку концептов для reshetka_elems.

Временная сложность алгоритма построения решетки концептов = $O(n^5)$

2.7 Код программы

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

vector<vector <int> > fm;
vector<vector <int> > levels_const;
vector<vector <int> > Z_fG;
vector <int> delete_el;
vector<vector <int> > m_p;
vector<vector <int> > Z_fG_copy;
vector <vector<vector <int> > > reshetka_elems;
vector <pair<int, int> > relations;
vector <int> min_res;
vector <int> max_res;
//vector <vector <pair<vector <int>, vector <int>> > > relations;
vector <int> mins;
```

```
vector <int> dop_l;
vector <int> dop_vec;
vector <int> delete_elem;
int help_lvl = 0;
int helper = -2; // отслеживание уровня диаграммы
void find_min(vector <int> a_has)  // определение ф-ии
{
mins.resize(0);
delete_elem.resize(0);
for (int i = 0; i < a_has.size(); i++) {
bool prov = 0;
for (int j = 0; j < i; j++) {
if (a_has[i] % a_has[j] == 0)
{
prov = 1;
}
}
if (prov == 0) {
mins.push_back(a_has[i]);
delete_elem.push_back(i);
}
}
levels_const.push_back(mins);
helper++;
}
void func_hasse(int x, bool q, vector <int> a_has)  // определение ф-ии
{
vector <int> diagramm_el;

for (int i = 2; i <= x; i++) {
if (x % i == 0)
{
a_has.push_back(i);
}
```

```
}
cout << "Делители числа " << x << " : ";
if (q == 1) {
cout << "1 ";
}
for (int k = 0; k < a_has.size(); k++)
{
cout << a_has[k] << " ";
}
cout << endl;
if (q == 1) {
cout << "Минимальные элементы: 1" << endl;
cout << "Наименьший элемент: 1" << endl;
cout << "Максимальные элементы: " << x << endl;
cout << "Наибольший элемент: " << x << endl;
diagramm_el.push_back(1);
//diagramm.push_back(diagramm_el);
levels_const.push_back(diagramm_el);
helper++;
}
else {
cout << "Минимальные элементы: ";
find_min(a_has);
for (int k = 0; k < mins.size(); k++)
{
cout << mins[k] << " ";
}
cout << endl;
if (mins.size() == 1)
cout << "Наименьший элемент: " << mins[0];
else
cout << "Наименьшего элемента нет";
cout << endl;
cout << "Максимальные элементы: " << x << endl;
```

```
cout << "Наибольший элемент: " << x << endl;
}
//  int hlp = 0;
// int stop;
while (!a_has.empty()) {
if (q == 1 || levels_const.size() > 1)
{
for (int i = 0; i < mins.size(); i++) {
for (int j = 0; j < levels_const[helper].size(); j++) {
if (mins[i] % levels_const[helper][j] == 0)
{
relations.push_back(make_pair(levels_const[helper][j], mins[i]));
}
}
}
for (int i = delete_elem.size() - 1; i >= 0; i--) {
a_has.erase(a_has.begin() + delete_elem[i]);
}
}
else
{
for (int i = delete_elem.size() - 1; i >= 0; i--) {
a_has.erase(a_has.begin() + delete_elem[i]);
}
}
if (!a_has.empty()) {
find_min(a_has);
}
}

cout << "Диаграмма Хассе: " << endl;
for (int i = levels_const.size() - 1; i >= 0; i--)
{
for (int j = 0; j < levels_const[i].size(); j++) {
```

```
cout << levels_const[i][j] << " ";
}
cout << endl;
}

cout << "Связи: " << endl;
for (int i = 0; i < relations.size(); i++)
{
cout << "( " << relations[i].first << " -> " << relations[i].second
<< " ) " << endl;
}
}

void z_reflexive(int N, int** a)
{
for (int i = 0; i < N; i++)
{
a[i][i] = 1;
}
}

void z_sim(int N, int** a)
{
for (int i = 0; i < N; i++)
{
for (int j = 0; j < N; j++)
{
if (a[i][j] == 1)
a[j][i] = 1;
}
}
}

void z_tranz(int N, int** a)
```

```
{
for (int i = 0; i < N; i++)
for (int j = 0; j < N; j++)
for (int k = 0; k < N; k++)
if (a[j][k] == 1)
for (int d = 0; d < N; d++)
if (a[k][d] == 1)
a[j][d] = 1;

}

void z_build(int N, int** z_a, int vvod)
{

z_reflexive(N, z_a);
z_sim(N, z_a);
z_tranz(N, z_a);

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


void    bo_result(int N, int** a)
{
cout << "Построенное эквивалентное замыкание: " << endl;
z_build(N, a, 4);

}

void    fm_result(int N, int** a)
```

```
{
for (int i = 0; i < N; i++) {
vector <int> vec;
fm.push_back(vec);
for (int j = 0; j < N; j++) {
if (a[i][j] == 1)
{
fm[fm.size() - 1].push_back(j + 1);
}
}
}
sort(fm.begin(), fm.end());
fm.resize(unique(fm.begin(), fm.end()) - fm.begin());

cout << endl;
cout << "Фактор множество:  ";
cout << "{ ";
for (int i = 0; i < fm.size(); i++) {
cout << "{";
for (int j = 0; j < fm[i].size(); j++)
{
cout << fm[i][j];
if (j != fm[i].size() - 1)
cout << ", ";
}
if (i == fm.size() - 1)
cout <<"} ";
else
cout << "}, ";
}
cout << "}" << endl;
cout << "Полная система представителей T =  ";
cout << "{ ";
for (int i = 0; i < fm.size(); i++) {
```

```
cout << fm[i][0];
if (i != fm.size() - 1)
cout << ", ";
}
cout << " }";
}

void min_m_p(int** a, int N) {
int** a_new = new int* [N];
for (int i = 0; i < N; i++) {
a_new[i] = new int[N];
for (int j = 0; j < N; j++)
a_new[i][j] = a[i][j];
}

bool prov = 1;
int min_el;
for (int k = 0; k < N; k++) {
min_el = 2 * N;
dop_l.resize(0);
for (int i = 0; i < N; i++) {
prov = true;
int sk = 0;
for (int j = 0; j < N; j++) {
if (a_new[j][i] == 2) {
prov = 0;
break;
}
if (a_new[j][i] == 1)
sk++;
}
if (prov) {
if (sk == min_el)
dop_l.push_back(i + 1);
```

```
else if (sk < min_el) {
min_el = sk;
dop_l.resize(0);
dop_l.push_back(i + 1);
}
}
}
k--;
for (int i = 0; i < dop_l.size(); i++) {
for (int j = 0; j < N; j++)
a_new[j][dop_l[i] - 1] = 2;
k++;
}

m_p.push_back(dop_l);
}

for (int i = 0; i < m_p[0].size(); i++)
min_res.push_back(m_p[0][i]);
}

void find_max(int** a) {

for (int i = 0; i < m_p.size() - 1; i++)
for (int j = 0; j < m_p[i].size(); j++) {
bool prov = true;
for (int k = 0; k < m_p[i + 1].size(); k++)
if (a[m_p[i][j] - 1][m_p[i + 1][k] - 1] == 1) {
prov = false;
break;
}

if (prov)
max_res.push_back(m_p[i][j]);
```

```
}

for (int i = 0; i < m_p[m_p.size() - 1].size(); i++)
max_res.push_back(m_p[m_p.size() - 1][i]);
}

// m_p v v
void    matrix_poryad(int N, int** a)
{
min_m_p(a, N);

cout << endl;
find_max(a);
cout << "Минимальные элементы: ";
for (int i = 0; i < min_res.size(); i++) {
if (i == min_res.size() - 1)
cout << min_res[i] << endl;
else
cout << min_res[i] << ", ";
}
if (min_res.size() > 1)
cout << "Наименьшего элемента нет " << endl;
else
cout << "Наименьший элемент: " << min_res[0] << endl;

cout << "Максимальные элементы: ";
for (int i = 0; i < max_res.size(); i++) {
if (i == max_res.size() - 1)
cout << max_res[i] << endl;
else
cout << max_res[i] << ", ";
}

if (max_res.size() > 1)
```

```
cout << "Наибольшего элемента нет " << endl;
else
cout << "Наибольший элемент: " << max_res[0] << endl;


cout << endl;
cout << "Диаграмма Хассе: " << endl;
for (int n = m_p.size() - 1; n >= 0; n--) {
for (int i = 0; i < m_p[n].size(); i++) {
if (i == m_p[n].size() - 1)
cout << m_p[n][i] << endl;
if (i != m_p[n].size() - 1)
cout << m_p[n][i] << " ";
}
}
cout << endl;

cout << "Связи в диаграмме Хассе" << endl;
for (int n = 0; n < m_p.size() - 1; n++) {
for (int i = 0; i < m_p[n].size(); i++) {
for (int j = 0; j < m_p[n + 1].size(); j++) {
if (a[m_p[n][i] - 1][m_p[n + 1][j] - 1] == 1) {
cout << "( " << m_p[n][i] << " -> " << m_p[n + 1][j] << " )" << endl;
}
}
}
}
}



// Z_fG vector vectorov
void    sist_zam(int N, vector <int> G_context,
 vector <char> M_context, int** matr)
```

```
{
Z_fG.push_back(G_context);
vector <int> rho_helper;
vector <int> intersection;
//строим систему замыканий
for (int i = 0; i < N; i++)
{
rho_helper.resize(0);
for (int j = 0; j < N; j++) {
if (matr[j][i] == 1) {
rho_helper.push_back(j + 1);
}
}
Z_fG.push_back(rho_helper);
int dop_size = Z_fG.size();
for (int j = 0; j < dop_size; j++) {
intersection.resize(0);
for (int k = 0; k < Z_fG[j].size(); k++) {
for (int l = 0; l < rho_helper.size(); l++) {
if (rho_helper[l] == Z_fG[j][k])
{
intersection.push_back(rho_helper[l]);
}
}
}
Z_fG.push_back(intersection);

}
}
sort(Z_fG.begin(), Z_fG.end());
Z_fG.resize(unique(Z_fG.begin(), Z_fG.end()) - Z_fG.begin());
//вывод системы замыканий
cout << "Система замыканий: ";
cout << endl;
```

```
cout << "Z_fG = { ";
for (int k = 0; k < Z_fG.size(); k++) {
cout << "{ ";
for (int l = 0; l < Z_fG[k].size(); l++) {
cout << Z_fG[k][l];
if (l != Z_fG[k].size() - 1)
{
cout << ", ";
}
}
if (k != Z_fG.size() - 1)
cout << " },";
else
cout << " }";
}
cout << " }";

}



void    reshetka_min(vector<vector <int> >& Z_fG_copy)
{
int hlp = 0;
bool flag1 = 0;
vector <int> in_vect;
vector<vector <int> > resh_min_elems;
resh_min_elems.resize(0);
in_vect.resize(0);
for (int i = Z_fG_copy.size() - 1; i >= 0; i--)
// i - рассматриваемый вектор
{

bool stop_p = 0;
bool flag = 0;
```

```
int real = 0;
in_vect.resize(0);
dop_vec.resize(0);
for (int j = 0; j < Z_fG_copy.size(); j++) // j - остальные векторы
{
real = 0;
for (int k = 0; k < Z_fG_copy[i].size(); k++) // k - свои элементы
{
for (int l = 0; l < Z_fG_copy[j].size(); l++)
// l - элементы другого вектора
{
if (i != j) {
if (Z_fG_copy[i][k] == Z_fG_copy[j][l]) {
in_vect.push_back(Z_fG_copy[i][k]);
}
}
}
}
if (in_vect.size() != Z_fG_copy[i].size() && i != j)
{
for (int h = 0; h < Z_fG_copy[j].size(); h++)
{
for (int t = 0; t < in_vect.size(); t++)
{
if (Z_fG_copy[j][h] == in_vect[t])
{
real++;
}
}
}
if (real == Z_fG_copy[j].size())
{
stop_p = 1;
}
```

```
}
if (stop_p == 1)
break;
in_vect.resize(0);
}
if (stop_p != 1)
{
resh_min_elems.push_back(Z_fG_copy[i]);
delete_el.push_back(i);
}
}
for (int ds = 0; ds < delete_el.size(); ds++)
{
Z_fG_copy.erase(Z_fG_copy.begin() + delete_el[ds]);
}
delete_el.resize(0);
reshetka_elems.push_back(resh_min_elems);
}



void    reshetka_pairs(vector < vector<vector <int> > >& reshetka_elems,
 int** matr, vector <char> M_context, vector <int> G_context, int G_M)
{
vector <char> char_elems;
vector < vector  <char> > vec_char_elems;
cout << "Итоговая решетка: ";
cout << endl;
for (int l = reshetka_elems.size() - 1; l >= 0; l--) {
for (int u = 0; u < reshetka_elems[l].size(); u++) {
vec_char_elems.resize(0);
char_elems.resize(0);
for (int i = 0; i < G_M; i++) {
// 3 и 4 не читаются, т.к. смотрятся первые 2 элемента
```

```
for (int q = 0; q < reshetka_elems[l][u].size(); q++) {
if (i == reshetka_elems[l][u][q] - 1) {
char_elems.resize(0);
for (int j = 0; j < G_M; j++) {
if (matr[i][j] == 1) {
char_elems.push_back(M_context[j]);
}
}
vec_char_elems.push_back(char_elems);
char_elems.resize(0);
if (vec_char_elems.size() > 1)
{
char_elems.resize(0);
for (int el_1 = 0; el_1 < vec_char_elems[0].size(); el_1++)
{
for (int el_2 = 0; el_2 < vec_char_elems[1].size(); el_2++)
{
if (vec_char_elems[0][el_1] == vec_char_elems[1][el_2]) {
char_elems.push_back(vec_char_elems[0][el_1]);
}
}
}
vec_char_elems.resize(0);
vec_char_elems.push_back(char_elems);
}
}
}
}
cout << " { { ";
if (reshetka_elems[l][u].size() == 0) {
cout << "}, {";
for (int dp = 0; dp < M_context.size(); dp++)
{
cout << M_context[dp];
```

```
if (dp != M_context.size() - 1)
cout << ", ";
}
cout << " } } ";
}
else if (reshetka_elems[l][u].size() == G_context.size()) {
for (int dp = 0; dp < G_context.size(); dp++)
{
cout << G_context[dp];
if (dp != G_context.size() - 1)
cout << ", ";
}
cout << " }, { } } ";
}
else {
for (int dp = 0; dp < reshetka_elems[l][u].size(); dp++)
{
cout << reshetka_elems[l][u][dp];
if (dp != reshetka_elems[l][u].size() - 1)
cout << ", ";
}
cout << " }, { ";
for (int dp = 0; dp < vec_char_elems[0].size(); dp++)
{
cout << vec_char_elems[0][dp];
if (dp != vec_char_elems[0].size() - 1)
cout << ", ";
}
cout << " } }";
}
}
cout << endl;
}
}
```

```
void    reshetka(vector<vector <int> > Z_fG, int** matr,
 vector <char> M_context, vector <int> G_context, int G_M)
{
//создаем копию
Z_fG_copy.resize(Z_fG.size());
for (int k = 0; k < Z_fG.size(); k++) {
Z_fG_copy[k].resize(Z_fG[k].size());
for (int l = 0; l < Z_fG[k].size(); l++) {
Z_fG_copy[k][l] = Z_fG[k][l];
}
}
for (int y = 0; y < Z_fG_copy.size(); y++)
{
if (Z_fG_copy[y].size() == 0)
{
reshetka_elems.resize(1);
reshetka_elems[0].push_back(dop_vec);
help_lvl++;
Z_fG_copy.erase(Z_fG_copy.begin() + y);
}
}
while (!Z_fG_copy.empty())
{
reshetka_min(Z_fG_copy);
}
//вывод диаграммы Хассе
cout << endl;
cout << "Диаграмма Хассе: " << endl;
for (int k = reshetka_elems.size() - 1; k >= 0; k--) {
for (int l = 0; l < reshetka_elems[k].size(); l++) {
cout << "{ ";
for (int u = 0; u < reshetka_elems[k][l].size(); u++) {
```

```
cout << reshetka_elems[k][l][u] << " ";
}
cout << " }";
}
cout << endl;
}
cout << endl;

//связи
cout << "Связи в диаграмме Хассе: " << endl;
int dop_r = 0;
for (int k = 0; k < reshetka_elems.size() - 1; k++) {
for (int l = 0; l < reshetka_elems[k].size(); l++) {
dop_r = 0;
for (int l2 = 0; l2 < reshetka_elems[k + 1].size(); l2++) {
dop_r = 0;
for (int u = 0; u < reshetka_elems[k][l].size(); u++) {
for (int c = 0; c < reshetka_elems[k + 1][l2].size(); c++) {
if (reshetka_elems[k][l][u] == reshetka_elems[k + 1][l2][c])
{
dop_r++;
}
}
}
if (dop_r == reshetka_elems[k][l].size())
{
cout << "( { ";
for (int s = 0; s < reshetka_elems[k][l].size(); s++)
{
cout << reshetka_elems[k][l][s] << " ";
}
cout << " } -> { ";
for (int s = 0; s < reshetka_elems[k + 1][l2].size(); s++)
{
```

```
cout << reshetka_elems[k + 1][12][s] << " ";
}
cout << " } )";
}
cout << endl;
}
cout << endl;
}

cout << endl;
}
//ДАЛЬШЕ
reshetka_pairs(reshetka_elems, matr, M_context, G_context, G_M);
}

bool bo_is_transitive(int N, int** a)
{
bool res = 0;

for (int i = 0; i < N; ++i)
{
for (int j = 0; j < N; ++j)
{
for (int k = 0; k < N; ++k)
{
if (a[i][j] >= a[i][k] * a[k][j])
res = 1;
else res = 0;

if (res == 0)
{
return  res;
}
}
```

```
}
}

return  res;
}

bool bo_is_antisymmetric(int N, int** a)
{
bool res = 0;

for (int i = 0; i < N; ++i)
{
for (int j = i + 1; j < N; ++j)
{
if (a[i][j] == 1 && a[j][i] == 1) {
if (i == j)
res = 1;
else res = 0;
}
else res = 1;

if (res == 0)
{
return  res;
}
}
}

return  res;
}

bool bo_is_reflexive(int N, int** a)
{
bool res = 0;
```

```
for (int i = 0; i < N; ++i)
{
if (a[i][i] == 1)
res = 1;
else res = 0;

if (res == 0)
{
return  res;
}
}

return  res;
}

int main()
{
setlocale(LC_ALL, "Rus");

int sposob, i, j, N, ch;
bool q1, q2;
int q_vibor;
cout << "Выберите действие: " << endl;
cout << "0 - построение эквивалентного замыкания бинарного отношения
и системы представителей фактор-множества" << endl;
cout << "1 - вычисление минимальных (максимальных) и наименьших
 (наибольших) элементов  и построение диаграммы Хассе " << endl;
cout << "2 - вычисление решетки концептов " << endl;
cin >> q_vibor;
if (q_vibor == 1) { // ВВЕЛИ 1
cout << "1 - ввод числом, 0 - ввод матрицей: ";
cin >> q1;
if (q1 == 1)
```

```
{
cout << "Надо ли включать единицу? 1 - да, 0 - нет: ";
cin >> q2;
cout << "Введите число: ";
cin >> ch;
vector <int> a_has;
func_hasse(ch, q2, a_has);
}
else
{
cout << "Введите размерность матрицы бинарного отношения: ";
cin >> N;
if (N == 0) {
cout << "Ошибка";
return 0;
}
int** a;
a = new int* [N];
cout << "Введите матрицу A" << endl;
for (int i = 0; i < N; i++) {
a[i] = new int[N];
for (int j = 0; j < N; j++) {
cin >> a[i][j];
}
}
int res_refl = bo_is_reflexive(N, a);
int res_antisimm = bo_is_antisymmetric(N, a);
int res_tranz = bo_is_transitive(N, a);
if (res_refl == 1 && res_antisimm == 1 && res_tranz == 1) {
cout << "Данное отношение является отношением порядка" << endl;
}
else
cout << "Данное отношение НЕ является отношением порядка" << endl;
```

```
matrix_poryad(N, a);
}
}
else if (q_vibor == 0) { //ВВЕЛИ 0
cout << "Введите размерность матрицы: ";
cin >> N;
if (N == 0) {
cout << "Ошибка";
return 0;
}
int** a;
a = new int* [N];
cout << "Введите матрицу A" << endl;
for (int i = 0; i < N; i++) {
a[i] = new int[N];
for (int j = 0; j < N; j++) {
cin >> a[i][j];
}
}
//построение замыкания эквивалентности
bo_result(N, a);
//система представителей фактор-множества
fm_result(N, a);
}
else { // ВВЕЛИ 2
int G_M;
cout << "Введите размеры множеств G и M: ";
cin >> G_M;
cout << "Введите множество объектов G: ";
vector <int> G_context;
for (int j = 0; j < G_M; j++) {
int x;
cin >> x;
G_context.push_back(x);
```

```
}
cout << "Введите множество атрибутов M: ";
vector <char> M_context;
for (int j = 0; j < G_M; j++) {
char x;
cin >> x;
M_context.push_back(x);
}
int** matr;
matr = new int* [G_M];
cout << "Введите матрицу: " << endl;
for (int i = 0; i < G_M; i++) {
matr[i] = new int[G_M];
for (int j = 0; j < G_M; j++) {
cin >> matr[i][j];
}
}
sist_zam(G_M, G_context, M_context, matr);
reshetka(Z_fG, matr, M_context, G_context, G_M);
}
}
```

2.8 Результаты тестирования программ

Тестирование №1:

Построение эквивалентного замыкания и системы представителей фактор-множества

```
Выберите действие:
0 - построение эквивалентного замыкания бинарного отношения и системы представителей фактор-множества
1 - вычисление минимальных (максимальных) и наименьших (наибольших) элементов  и построение диаграммы Хассе
2 - вычисление решетки концептов
0
Введите размерность матрицы: 4
Введите матрицу А
0 1 0 0
1 1 0 0
0 0 0 1
0 0 0 1
Построенное эквивалентное замыкание:
1 1 0 0
1 1 0 0
0 0 1 1
0 0 1 1

Фактор множество:  { {1, 2}, {3, 4} }
Полная система представителей T =  { 1, 3 }
```

Рисунок 1 – Тестировние №1

Тестирование №2:

Вычисление минимальных (максимальных) и наименьших (наибольших) элементов и построение диаграммы Хассе. Ввод числом. Единица включается.

```
Выберите действие:
0 - построение эквивалентного замыкания бинарного отношения и системы представителей фактор-множества
1 - вычисление минимальных (максимальных) и наименьших (наибольших) элементов  и построение диаграммы Хассе
2 - вычисление решетки концептов
1
1 - ввод числом, 0 - ввод матрицей: 1
Надо ли включать единицу? 1 - да, 0 - нет: 1
Введите число: 45
Делители числа 45 : 1 3 5 9 15 45
Минимальные элементы: 1
Наименьший элемент: 1
Максимальные элементы: 45
Наибольший элемент: 45
Диаграмма Хассе:
45
9 15
3 5
1
Связи:
( 1 -> 3 )
( 1 -> 5 )
( 3 -> 9 )
( 3 -> 15 )
( 5 -> 15 )
( 9 -> 45 )
( 15 -> 45 )
```

Рисунок 2 – Тестировние №2

Тестирование №3:

Вычисление минимальных (максимальных) и наименьших (наибольших) элементов и построение диаграммы Хассе. Ввод числом. Единица не включается.

```
Выберите действие:
0 - построение эквивалентного замыкания бинарного отношения и системы представителей фактор-множества
1 - вычисление минимальных (максимальных) и наименьших (наибольших) элементов  и построение диаграммы Хассе
2 - вычисление решетки концептов
1
1 - ввод числом, 0 - ввод матрицей: 1
Надо ли включать единицу? 1 - да, 0 - нет: 0
Введите число: 62
Делители числа 62 : 2 31 62
Минимальные элементы: 2 31
Наименьшего элемента нет
Максимальные элементы: 62
Наибольший элемент: 62
Диаграмма Хассе:
62
2 31
Связи:
( 2 -> 62 )
( 31 -> 62 )
```

Рисунок 3 – Тестировние №3

Тестирование №4:

Вычисление минимальных (максимальных) и наименьших (наибольших) элементов и построение диаграммы Хассе. Ввод матрицей.

```
Выберите действие:
0 - построение эквивалентного замыкания бинарного отношения и системы представителей фактор-множества
1 - вычисление минимальных (максимальных) и наименьших (наибольших) элементов  и построение диаграммы Хассе
2 - вычисление решетки концептов
1
1 - ввод числом, 0 - ввод матрицей: 0
Введите размерность матрицы бинарного отношения: 4
Введите матрицу A
1 1 1 1
0 1 1 1
0 0 1 1
0 0 0 1
Данное отношение является отношением порядка

Минимальные элементы: 1
Наименьший элемент: 1
Максимальные элементы: 4
Наибольший элемент: 4

Диаграмма Хассе:
4
3
2
1

Связи в диаграмме Хассе
( 1 -> 2 )
( 2 -> 3 )
( 3 -> 4 )
```

Рисунок 4 – Тестировние №4

Тестирование №5:

Построение решетки концептов.

```
Консоль отладки Microsoft Visual Studio
1 - вычисление минимальных (максимальных) и наименьших (наибольших) элементов  и построение диаграммы Хассе
2 - вычисление решетки концептов
2
Введите размеры множеств G и M: 4
Введите множество объектов G: 1 2 3 4
Введите множество атрибутов M: a b c d
Введите матрицу:
0 1 0 0
1 0 1 0
0 0 1 0
0 0 1 1
Система замыканий:
Z_fG = { {  },{ 1 },{ 1, 2, 3, 4 },{ 2 },{ 2, 3, 4 },{ 4 } }
Диаграмма Хассе:
{ 1 2 3 4  }
{ 2 3 4  }
{ 4  }{ 2  }{ 1  }
{  }

Связи в диаграмме Хассе:
( {  } -> { 4  } )
( {  } -> { 2  } )
( {  } -> { 1  } )


( { 4  } -> { 2 3 4  } )

( { 2  } -> { 2 3 4  } )




( { 2 3 4  } -> { 1 2 3 4  } )


Итоговая решетка:
 { { 1, 2, 3, 4 }, { } }
 { { 2, 3, 4 }, { c } }
 { { 4 }, { c, d } } { { 2 }, { a, c } } { { 1 }, { b } }
 { { }, {a, b, c, d } }
```

Рисунок 5 – Тестировние №5

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были рассмотрены и изучены следующие темы: определения отношения эквивалентности, фактор-множества, определения отношения порядка и диаграммы Хассе, определения контекста и концепта. В третьей части работы были реализованы алгоритмы построения эквивалентного замыкания бинарного отношения и системы представителей фактор-множества, алгоритмы вычисления минимальных (максимальных) и наименьших (наибольших) элементов и построения диаграммы Хассе и алгоритм вычисления решетки концептов.