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

Норикова Павла Сергеевича

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

Саратов 2022

ВВЕДЕНИЕ

Цель данной работы — изучение основных свойств бинарных отношений и операций замыкания бинарных отношений.

ОТНОШЕНИЕ ЭКВИВАЛЕНТНОСТИ

Бинарное отношение *ρ* на множестве *M* называется рефлексивным, если для любого элемента *a* из *M*, выполняется условие *a* *ρ* *a*.

∀ *a*∈*M*  *a* *ρ* *a* или ∀ *a*∈*M*  (*a*,*a*)∈*ρ*

Бинарное отношение *ρ* на множестве *M* называется симметричным, если для любых двух элементов *a*,*b* из *M*, из условия *a* *ρ* *b* следует условие *b* *ρ* *a*

∀*a*,*b*∈*M*  *a* *ρ* *b* → *b* *ρ* *a* или

∀ *a*,*b*∈*M*  (*a*,*b*)∈*ρ* → (*b*,*a*)∈*ρ*

Бинарное отношение *ρ* на множестве *M* называется антисимметричным, если для любых элементов *a*,*b* из *M*, из условий *a* *ρ* *b* и *b* *ρ* *a* следует условие *a*=*b*

∀ *a*,*b*,*c*∈*M*  *a* *ρ* *b* ∧ *b* *ρ* *a* → *a*=*b* или

∀ *a*,*b*∈*M*  (*a*,*b*)∈*ρ* ∧ (*b*,*a*)∈*ρ* → *a*=*b*

Бинарное отношение *ρ* на множестве *M* называется транзитивным, если для любых элементов *a*,*b*,*c* из *M*, из условий *a* *ρ* *b* и *b* *ρ* *c* следует условие *a* *ρ* *c*

∀ *a*,*b*,*c*∈*M*  *a* *ρ* *b* ∧ *b* *ρ* *c* → *a* *ρ* *c* или

∀ *a*,*b*,*c*∈*M*  (*a*,*b*)∈*ρ* ∧ (*b*,*c*)∈*ρ* → (*a*,*c*)∈*ρ*

Бинарное отношение ρ на множестве M называется отношением эквивалентности, если оно рефлексивно, симметрично и транзитивно.

Алгоритм 1 – построение замыкания рефлексивности:

*Вход.* Матрица бинарного отношения v, количество элементов множества N.

*Выход.* Матрица замыкания рефлексивности v2.

Шаг 1. Создаём матрицу v2. Переносим в неё все элементы матрицы v. В цикле по i от 0 до N - 1 присваиваем значениям v2[i][i] единицу.

Шаг 2. Выводим матрицу v2.

Алгоритм 2 - построение замыкания симметричности:

*Вход.* Матрица бинарного отношения v, количество элементов множества N.

*Выход.* Матрица замыкания симметричности v2.

Шаг 1. Создаём матрицу v2. Переносим в неё все элементы матрицы v. Проверяем все значения v2[i][j], i,j от 0 до N - 1. Если значение v2[i][j] равны единице, присваиваем значениям v2[j][i] единицу.

Шаг 2. Выводим матрицу v2.

Алгоритм 3 - построение замыкания транзитивности:

*Вход.* Матрица бинарного отношения v, количество элементов множества N.

*Выход.* Матрица замыкания транзитивности v2.

Шаг 1. Создаём матрицу v2. Переносим в неё все элементы матрицы v. Проверяем все значения v2[i][j], i,j от 0 до N - 1. Если значение оказалось равным единице, то для нашего j проверяем все значения v2[j][k], k от 0 до N - 1. Если для какого-либо k значение v2[j][k] равно единице, то присваиваем значению v2[i][k] единицу.

Шаг 2. Повторяем первый шаг N раз.

Шаг 3. Выводим матрицу v2.

Алгоритм 4 - построение замыкания эквивалентности:

*Вход.* Матрица бинарного отношения v, количество элементов множества N.

*Выход.* Матрица замыкания эквивалентности v2.

Шаг 1. Создаём матрицу v2. Переносим в неё все элементы матрицы v. Для матрицы v2 применяем последовательно алгоритмы 9, 10, 11.

Шаг 2. Выводим матрицу v2.

ФАКТОР-МНОЖЕСТВО

Срезы ε(а) называются классами эквивалентности по отношению ε и сокращенно обозначаются символом [a].

Множество всех таких классов эквивалентности {[а]:а ∈ A} называется фактор-множеством множества А по эквивалентности ε и обозначается символом A/ε.

Подмножество T⊂A называется полной системой представителей классов эквивалентности ε на множестве A, если:

1) ε(T) = A,

2) из условия t1 = t2(ε) следует t1 = t2.

Классы эквивалентности [t] ∈ А/ε могут быть отождествлены со своими представителями t и фактор-множество A/ε может быть отождествлено с множеством T.

Алгоритм 5 – построение фактор-множества:

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

*Выход.* Фактор-множество v2 эквивалентного отношения v.

Шаг 1. Создаём список списков v2 и массив посещённых вершин used.

Шаг 2. Для каждого элемента i ∈ A проверяем used[i].

Шаг 2.1. Если i ещё не посещён, помечаем i как посещённый, добавляем i в новый список x\_i.

Шаг 2.2. Создаём список x\_i и для всех j ∈ A проверяем v[i][j].

Шаг 2.2.1. Если v[i][j] = 1 и j – не посещён, помечаем j как посещённый и добавляем j в x\_i.

Шаг 2.3. Кладём список x\_i в v2.

Шаг 3. Выводим фактор-множество v2.

Алгоритм 6 – построение системы представителей фактор-множества:

*Вход.* Фактор-множество v.

*Выход.* Система представителей v2 фактор-множества v.

Шаг 1. Для каждого среза из фактор-множества v

Шаг 1.1. Возьмём его наименьший элемент x.

Шаг 1.2. Добавим x в v2.

Шаг 2. Выводим фактор-множество v2.

ОТНОШЕНИЕ ПОРЯДКА И ДИАГРАММА ХАССЕ

Бинарное отношение ω на множестве А называют отношением порядка, если оно рефлексивно, антисимметрично и транзитивно.

Поскольку отношение порядка интуитивно отражает свойство «больше-меньше», то для обозначения порядка ω используется инфиксная запись с помощью символа ≤: вместо (a, b) ∈ ω принято писать a ≤ b.

Запись а < b означает, что а ≤ b и а ≠ b.

Запись а <**•** b означает, что а ≤ b и нет элементов х, удовлетворяющих условию а < х < b. В этом случае говорят, что элемент b покрывает элемент а или что элемент а покрывается элементом b.

Элементы а, b ∈ А называются сравнимыми, если а ≤ b или b ≤ а, и несравнимыми в противном случае.

Множество А с заданным на нём отношением порядка ≤ называется упорядоченным множеством и обозначается А = (А, ≤) или просто (А, ≤).

Упорядоченное множество А = (А, ≤) наглядно представляется диаграммой Хассе, которая представляет элементы множества А точками плоскости и пары а <**•** b представляет линиями, идущими вверх от элемента а к элементу b.

Элемент а упорядоченного множества (А, ≤) называется:

- минимальным, если (∀x ∈ A) x ≤ a ⇒ x = a,

- максимальным, если (∀x ∈ A) a ≤ x ⇒ x = a,

- наименьшим, если (∀x ∈ A) a ≤ x,

- наибольшим, если (∀x ∈ A) x ≤ a.

Лемма. Для любого конечного упорядоченного множества А = (А, ≤) справедливы следующие утверждения:

1) любой элемент множества А содержится в некотором максимальном элементе и содержит некоторый минимальный элемент;

2) если упорядоченное множество А имеет один максимальный (соответственно, минимальный) элемент, то он является наибольшим (соответственно, наименьшим) элементом этого множества.

Алгоритм 7 – построение диаграммы Хассе:

*Вход.* Конечное упорядоченное множество А = (А, ≤), обозначим v.

*Выход.* Диаграмма Хассе res.

Шаг 1. Заводим переменную lvl = 0 и список списков res.

Шаг 2. Для v находим список всех минимальных элементов, кладём его в res[lvl], удаляем элементы этого списка из v, lvl = lvl + 1.

Шаг 3. Повторяем Шаг 2 до тех пор, пока v – непустой.

Шаг 4. Создаём список пар pairs. Для всех уровней i от 0 до lvl – 2.

Шаг 4.1. Для всех j, j – элемент i-го уровня.

Шаг 4.1.1. Для всех k, k - элемент k-го уровня.

Шаг 4.1.1.1. Если res[i][j] ≤ res[i + 1][k], добавляем пару (res[i][j], res[i + 1][k]) в pairs.

Шаг 5. Выводим res и pairs.

Алгоритм 8 – поиск минимальных и наименьших элементов упорядоченного множества:

*Вход.* Конечное упорядоченное множество (А, ≤), обозначим v.

*Выход.* Список минимальных элементов res и наименьший элемент min.

Шаг 1. Заводим переменную min и список res.

Шаг 2. Для каждого i ∈ A

Шаг 2.1. Заводим переменную checker = 0 и для каждого j ∈ A

Шаг 2.1.1. Если v[j] ≤ v[i] и i != j, checker = checker + 1.

Шаг 2.2. Если checker = 0, кладём i в res.

Шаг 3. Если элементов в res больше одного, то наименьших элементов нет. Если в res только один элемент x, min = x.

Алгоритм 9 – поиск максимальных и наибольших элементов упорядоченного множества:

*Вход.* Конечное упорядоченное множество (А, ≤), обозначим v.

*Выход.* Список максимальных элементов res и наибольший элемент max.

Шаг 1. Заводим переменную max и список res.

Шаг 2. Для каждого i ∈ A

Шаг 2.1. Заводим переменную checker = 0 и для каждого j ∈ A

Шаг 2.1.1. Если v[i] ≤ v[j] и i != j, checker = checker + 1.

Шаг 2.2. Если checker = 0, кладём i в res.

Шаг 3. Если элементов в res больше одного, то наименьших элементов нет. Если в res только один элемент x, max = x.

КОНТЕКСТЫ И КОНЦЕПТЫ

Теорема Галуа. Пусть *ρ* ⊂ А×В - произвольное бинарное отношение между элементами множеств А и В. Для любого подмножества Х ⊂ А определим подмножество X' ⊂ В по формуле

Х' = {b ∈ B | (х, b) ∈ ρ для всех х ∈ Х}

и для любого подмножества Y ⊂ В определим подмножество Y' ⊂ А по формуле

Y’ = {a ∈ А| (а, y) ∈ ρ для всех у ∈ Y}.

Тогда отображения Х → Х' и Y → Y' множеств Р(А) и Р(В) друг в друга обладают следующими свойствами:

1) Х1 ⊂ X2 ⇒ X2' ⊂ X1';

2) Y1 ⊂ Y2 ⇒ Y2' ⊂ Y1';

3) X ⊂ X'', Y ⊂ Y'';

4) Х"' = Х' , Y"' = Y'.

Пара таких отображений называется соответствием Галуа, которое определяет два оператора замыкания fA и fB на множествах А и В по формулам:

fA(X) = Х'' для любых X ⊂ А и fB(Y) = Y'' для любых Y ⊂ B.

При этом отображения Х → Х' и Y → Y' определяют антиизоморфизмы между системами замыканий и на множествах А и В, т.е. это биекции, которые удовлетворяют условиям:

X ⊂ Y ⇔ X' ⊃ Y' для любых Х,Y ∈ ,

X ⊂ Y ⇔ X' ⊃ Y' для любых Х,Y ∈ .

Контекстом называется алгебраическая система K = (G, M, ρ), состоящая из множества объектов G, множества атрибутов M и бинарного отношения *ρ* ⊂ G × M, (g, m) ∈ *ρ* показывает, что объект g имеет атрибут m. Где g ∈ G, m ∈ M.

Упорядоченная пара (X, Y) замкнутых множеств X ∈ , Y ∈ , удовлетворяющих условиям φ(X) = Y, ψ(Y) = X, называется концептом контекста K = (G, M, ρ). При этом компонента X называется объёмом и компонента Y – содержанием концепта (X, Y).

Множество всех концептов *C*(*K*) так упорядочивается отношением (X, Y) ≤ (X1, Y1) ⇔ X  X1 (или равносильно Y1  Y) что (*C*(*K*), ≤) является полной решёткой, которая изоморфна решётке замкнутых подмножеств множества *G*.

В частности, по определению для g ∈ G, m ∈ M получаем

φ({g}) = {m ∈ M | (x, m) ∈ ρ для всех объектов x ∈ {g}} = ρ(g),

ψ({m}) = {g ∈ G | (g, y) ∈ ρ для всех атрибутов y ∈ {m}} = ρ-1(m).

Тогда для X  G, Y  M получаем

φ(X) = = , ψ(Y) = =

fG(X) = X'' = = = ,

M() = = = = .

Алгоритм 10 – вычисление системы замыканий :

*Вход.* Контекст K = (G, M, ρ).

*Выход.* Система замыканий .

Шаг 1. Заводим список списков res и кладём туда G.

Шаг 2. Для каждого i ∈ M

Шаг 2.1. Заводим список count и для каждого j ∈ G

Шаг 2.1.1. Если v[j][i] = 1 кладём j в count.

Шаг 2.2. Если в списке res ещё нет списка count, кладём count в res.

Шаг 2.3. Пересекаем count со всеми списками из res. Если пересечения ещё нет в списке res, кладём пересечение в res.

Шаг 5. Выводим res.

Алгоритм 11 – вычисление решётки концептов С(K):

*Вход.* Контекст K = (G, M, ρ).

*Выход.* Решётка концептов C(K).

Шаг 1. Заводим список списков пар res. Вычисляем систему замыканий по алгоритму 10.

Шаг 2. Для каждого подмножества i  рассматриваем φ(ji) для всех ji ∈ i.

Шаг 2.1. Пересекаем все φ(ji) для всех ji ∈ i и ставим это пересечение в пару с i.

Шаг 2.2. Добавляем эту пару в res.

Шаг 3. Выводим res.

ВРЕМЕННАЯ СЛОЖНОСТЬ АЛГОРИТМОВ

Построение замыкания рефлексивности - O(n)

Построение замыкания симметричности - O(n2)

Построение замыкания транзитивности - O(n4)

Построение замыкания эквивалентности - O(n4)

Построение фактор-множества – O(n2)

Построение системы представителей фактор-множества при условии, что элементы классов эквивалентности упорядочены – O(n)

Построение системы представителей фактор-множества при условии, что элементы классов эквивалентности не упорядочены – O(n3)

Построение диаграммы Хассе - O(n4)

Поиск минимальных и наименьших элементов упорядоченного множества - O(n2)

Поиск максимальных и наибольших элементов упорядоченного множества - O(n2)

Вычисление системы замыканий - O(n4)

Вычисление решётки концептов - O(n5)

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

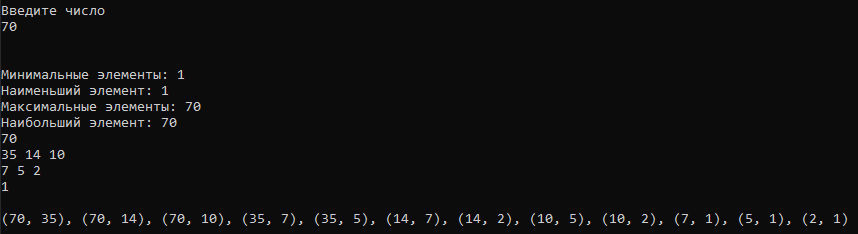


Рис.1 – первый тест

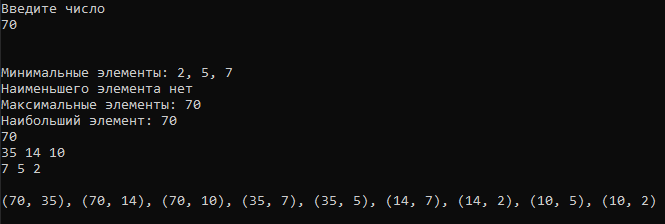


Рис.2 – второй тест

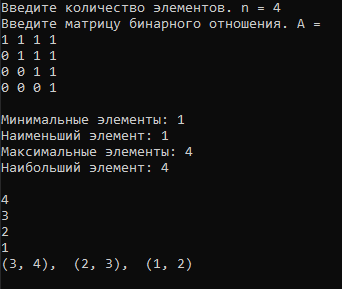


Рис.3 – третий тест

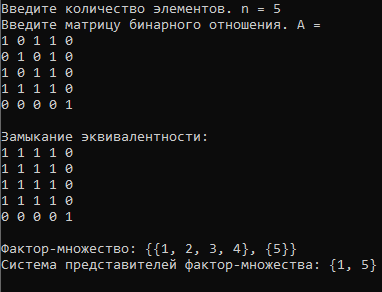


Рис.4 – четвёртый тест

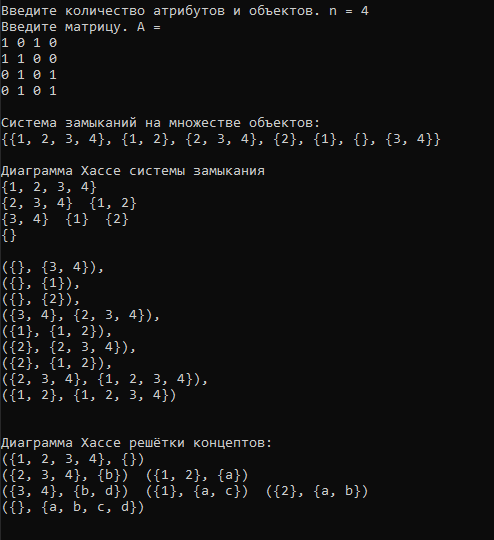


Рис.5 – пятый тест

КОД ПРОГРАММЫ

#include <iostream>

#include <vector>

using namespace std;

void make\_ref(int\*\* v, int n)

{

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

v[i][i] = 1;

}

void make\_sim(int\*\* v, int n)

{

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

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

if (v[i][j] == 1)

v[j][i] = 1;

}

void make\_tran(int\*\* v, int n)

{

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

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

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

if (v[j][k] == 1) {

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

if (v[k][p] == 1)

v[j][p] = 1;

}

}

}

}

}

}

void eq(int\*\* v, int n) {

int\*\* v2;

v2 = new int\* [n];

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

v2[i] = new int[n];

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

v2[i][j] = v[i][j];

}

}

make\_ref(v2, n);

make\_sim(v2, n);

make\_tran(v2, n);

cout << endl << "Замыкание эквивалентности:" << endl;

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

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

cout << v2[i][j] << ' ';

cout << endl;

}

vector<vector<int>> res;

int\* count = new int[n];

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

count[i] = 1;

}

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

if (count[i]) {

vector<int> srez;

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

if (v2[i][j] && count[j]) {

count[j] = 0;

srez.push\_back(j);

}

}

count[i] = 0;

res.push\_back(srez);

}

}

cout << endl << "Фактор-множество: {";

for (int i = 0; i < res.size(); i++) {

cout << "{";

for (int j = 0; j < res[i].size(); j++) {

cout << res[i][j] + 1;

if (j != res[i].size() - 1)

cout << ", ";

}

cout << "}";

if (i != res.size() - 1)

cout << ", ";

}

cout << "}" << endl;

cout << "Система представителей фактор-множества: {";

for (int i = 0; i < res.size(); i++) {

cout << res[i][0] + 1;

if (i != res.size() - 1)

cout << ", ";

}

cout << "}" << endl << endl;

}

vector <pair <int, int> > find\_all\_min(vector <int> v) {

vector <pair<int, int> > res;

res.push\_back(make\_pair(v[0], 0));

for (int i = 1; i < v.size(); i++) {

int checker = 0;

for (int j = 0; j < res.size(); j++)

if (v[i] % res[j].first == 0)

checker++;

if (checker == 0)

res.push\_back(make\_pair(v[i], i));

}

return res;

}

void hasse(vector <int> v) {

vector <vector <int>> res;

int lvl = 0;

while (v.size() > 0) {

vector <pair<int, int>> A\_i = find\_all\_min(v);

vector <int> help;

for (int i = A\_i.size() - 1; i >= 0; i--) {

v.erase(v.begin() + A\_i[i].second);

help.push\_back(A\_i[i].first);

}

res.push\_back(help);

lvl++;

}

for (int i = res.size() - 1; i >= 0; i--) {

for (int j = 0; j < res[i].size(); j++) {

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

}

cout << endl;

}

vector <pair<int, int>> res\_pairs;

for (int i = res.size() - 1; i > 0; i--) {

for (int j = 0; j < res[i].size(); j++) {

for (int k = 0; k < res[i - 1].size(); k++) {

if (res[i][j] % res[i - 1][k] == 0) {

res\_pairs.push\_back(make\_pair(res[i][j], res[i - 1][k]));

}

}

}

}

cout << endl;

for (int i = 0; i < res\_pairs.size(); i++) {

cout << '(' << res\_pairs[i].first << ", " << res\_pairs[i].second << ")";

if (i < res\_pairs.size() - 1) cout << ", ";

}

cout << endl;

}

vector <int> find\_all(int x, int one) {

vector <int> res;

for (int i = 2 - one; i < x / 2.0 + 1; i++) {

if (x % i == 0)

res.push\_back(i);

}

res.push\_back(x);

return res;

}

vector <int> find\_all\_min\_matr(vector <vector <int>> v, int n) {

vector <int> res;

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

if (v[i][i] > -1) {

int checker = 0;

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

if (v[j][i] == 1)

if (i != j)

checker++;

if (checker == 0)

res.push\_back(i);

}

}

return res;

}

vector <int> find\_all\_max\_matr(vector <vector <int>> v, int n) {

vector <int> res;

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

int checker = 0;

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

if (v[i][j])

if (i != j)

checker++;

if (checker == 0)

res.push\_back(i);

}

return res;

}

void hasse\_pair(vector <vector <int>> v, int n, vector <vector <int>> hasse\_diag) {

vector <pair<int, int>> res;

for (int i = hasse\_diag.size() - 1; i > 0; i--)

for (int j = 0; j < hasse\_diag[i].size(); j++)

for (int k = 0; k < hasse\_diag[i - 1].size(); k++)

if (v[hasse\_diag[i - 1][k]][hasse\_diag[i][j]])

res.push\_back(make\_pair(hasse\_diag[i - 1][k] + 1, hasse\_diag[i][j] + 1));

for (int i = 0; i < res.size(); i++) {

cout << "(" << res[i].first << ", " << res[i].second << ")";

if (i != res.size() - 1) cout << ", ";

else cout << endl << endl;

}

}

void hasse\_matr(vector <vector <int>> v\_real, int n) {

vector <vector <int>> v;

v.resize(n);

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

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

v[i].push\_back(v\_real[i][j]);

}

vector <vector <int>> res;

while (true) {

vector <int> A\_i = find\_all\_min\_matr(v, n);

if (A\_i.size() == 0)

break;

vector <int> help;

for (int k = A\_i.size() - 1; k >= 0; k--) {

for (int i = v.size() - 1; i >= 0; i--) {

for (int j = v[i].size() - 1; j >= 0; j--) {

if (A\_i[k] == j)

v[i][j] = -1;

}

if (A\_i[k] == i) {

for (int j = v[i].size() - 1; j >= 0; j--)

v[i][j] = -1;

}

}

help.push\_back(A\_i[k]);

}

res.push\_back(help);

}

for (int i = res.size() - 1; i >= 0; i--) {

for (int j = 0; j < res[i].size(); j++) {

cout << res[i][j] + 1 << " ";

}

cout << endl;

}

hasse\_pair(v\_real, n, res);

}

vector<int> p(vector<int> v1, vector<int> v2) {

vector<int> res;

for (int i = 0; i < v1.size(); i++)

for (int j = 0; j < v2.size(); j++)

if (v1[i] == v2[j])

res.push\_back(v1[i]);

return res;

}

bool checker(vector<vector<int>> v1, vector<int> v2) {

for (int i = 0; i < v1.size(); i++) {

if (v1[i].size() == v2.size()) {

int a = v2.size();

for (int j = 0; j < v1[i].size(); j++) {

if (v1[i][j] == v2[j])

a--;

}

if (a == 0)

return false;

}

}

return true;

}

vector <pair<vector <int>, int>> find\_all\_min\_zfg(vector<vector<int>> v) {

vector <pair<vector <int>, int>> res;

for (int i = 0; i < v.size(); i++) {

int count = 0;

for (int j = 0; j < v.size(); j++) {

if (p(v[i], v[j]) == v[i] || p(v[i], v[j]) != v[j]) {

count++;

}

}

if (count == v.size())

res.push\_back(make\_pair(v[i], i));

}

return res;

}

vector<int> fi(int\*\* matr, int x, int n) {

vector<int> res;

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

if (matr[x][i])

res.push\_back(i);

}

return res;

}

void c(vector<vector<vector<int>>> v, int\*\* matr, int n) {

char alf[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o' };

vector<vector<pair<vector<int>, vector<int>>>> res;

res.resize(v.size());

for (int i = 0; i < v.size(); i++) {

for (int j = 0; j < v[i].size(); j++) {

vector<int> count;

int checker = 0;

for (int k = 0; k < v[i][j].size(); k++) {

checker++;

if (k == 0)

count = fi(matr, v[i][j][k], n);

else

count = p(count, fi(matr, v[i][j][k], n));

}

if (checker > 0)

res[i].push\_back(make\_pair(v[i][j], count));

else {

vector<int> help;

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

help.push\_back(i);

res[i].push\_back(make\_pair(v[i][j], help));

}

}

}

cout << endl << "Диаграмма Хассе решётки концептов:" << endl;

for (int i = res.size() - 1; i >= 0; i--) {

for (int j = 0; j < res[i].size(); j++) {

cout << "({";

for (int k = 0; k < res[i][j].first.size(); k++) {

cout << res[i][j].first[k] + 1;

if (k != res[i][j].first.size() - 1) cout << ", ";

}

cout << "}, {";

for (int k = 0; k < res[i][j].second.size(); k++) {

cout << alf[res[i][j].second[k]];

if (k != res[i][j].second.size() - 1) cout << ", ";

}

cout << "}) ";

}

cout << endl;

}

}

void hasse\_zfg(vector<vector<int>> v, int\*\* matr, int n) {

vector<vector<vector<int>>> res;

int lvl = 0;

while (v.size() > 0) {

vector <pair<vector <int>, int>> A\_i = find\_all\_min\_zfg(v);

vector<vector<int>> help;

for (int i = A\_i.size() - 1; i >= 0; i--) {

v.erase(v.begin() + A\_i[i].second);

help.push\_back(A\_i[i].first);

}

res.push\_back(help);

lvl++;

}

cout << "Диаграмма Хассе системы замыкания " << endl;

for (int i = res.size() - 1; i >= 0; i--) {

for (int j = 0; j < res[i].size(); j++) {

cout << "{";

for (int k = 0; k < res[i][j].size(); k++) {

cout << res[i][j][k] + 1;

if (k != res[i][j].size() - 1) cout << ", ";

}

cout << "} ";

}

cout << endl;

}

vector<pair<vector<int>, vector<int>>> res\_pairs;

for (int i = 0; i < res.size() - 1; i++) {

for (int j = 0; j < res[i].size(); j++) {

for (int k = 0; k < res[i + 1].size(); k++) {

if (p(res[i][j], res[i + 1][k]) == res[i][j]) {

res\_pairs.push\_back(make\_pair(res[i][j], res[i + 1][k]));

}

}

}

}

cout << endl;

for (int i = 0; i < res\_pairs.size(); i++) {

cout << "({";

for (int j = 0; j < res\_pairs[i].first.size(); j++) {

cout << res\_pairs[i].first[j] + 1;

if (j != res\_pairs[i].first.size() - 1)

cout << ", ";

}

cout << "}";

cout << ", {";

for (int j = 0; j < res\_pairs[i].second.size(); j++) {

cout << res\_pairs[i].second[j] + 1;

if (j != res\_pairs[i].second.size() - 1)

cout << ", ";

}

cout << "}";

cout << ")";

if (i != res\_pairs.size() - 1)

cout << ", " << endl;

}

cout << endl << endl;

c(res, matr, n);

}

vector<vector<int>> zfg(int\*\* v, int n) {

vector<vector<int>> res;

vector<int> G;

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

G.push\_back(i);

res.push\_back(G);

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

vector<int> count;

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

if (v[j][i])

count.push\_back(j);

}

if (checker(res, count))

res.push\_back(count);

for (int i = 0; i < res.size(); i++) {

vector<int> new\_count = p(res[i], count);

if (checker(res, new\_count))

res.push\_back(new\_count);

}

}

cout << endl << "Система замыканий на множестве объектов:" << endl << "{";

for (int i = 0; i < res.size(); i++) {

cout << "{";

for (int j = 0; j < res[i].size(); j++) {

cout << res[i][j] + 1;

if (j != res[i].size() - 1)

cout << ", ";

}

cout << "}";

if (i != res.size() - 1)

cout << ", ";

}

cout << "}" << endl << endl;

hasse\_zfg(res, v, n);

return res;

}

int main()

{

setlocale(LC\_ALL, "Russian");

int h = -1;

while (h != 0)

{

cout << "1 - Порядок. Целое число, включая единицу" << endl;

cout << "2 - Порядок. Целое число, не включая единицу" << endl;

cout << "3 - Порядок. Матрица порядка " << endl;

cout << "4 - Построение эквивалентного замыкания бинарного отношения и системы представителей фактор-множества " << endl;

cout << "5 - Вычисление решетки концептов" << endl;

cout << "0 - Выход" << endl;

cin >> h;

if (h == 1) {

int x;

cout << "Введите число" << endl;

cin >> x;

cout << endl;

vector <int> res = find\_all(x, 1);

vector <pair<int, int>> min = find\_all\_min(res);

cout << endl << "Минимальные элементы: ";

for (int i = 0; i < min.size(); i++) {

cout << min[i].first;

if (i < min.size() - 1) cout << ", ";

}

cout << endl;

if (min.size() == 1) cout << "Наименьший элемент: " << min[0].first << endl;

else cout << "Наименьшего элемента нет" << endl;

cout << "Максимальные элементы: " << x << endl;

cout << "Наибольший элемент: " << x << endl;

hasse(res);

}

if (h == 2) {

int x;

cout << "Введите число" << endl;

cin >> x;

cout << endl;

vector <int> res = find\_all(x, 0);

vector <pair<int, int>> min = find\_all\_min(res);

cout << endl << "Минимальные элементы: ";

for (int i = 0; i < min.size(); i++) {

cout << min[i].first;

if (i < min.size() - 1) cout << ", ";

}

cout << endl;

if (min.size() == 1) cout << "Наименьший элемент: " << min[0].first << endl;

else cout << "Наименьшего элемента нет" << endl;

cout << "Максимальные элементы: " << x << endl;

cout << "Наибольший элемент: " << x << endl;

hasse(res);

}

if (h == 3) {

int n;

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

cin >> n;

vector <vector <int>> v;

v.resize(n);

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

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

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

int x\_cin;

cin >> x\_cin;

v[i].push\_back(x\_cin);

}

}

vector <int> min = find\_all\_min\_matr(v, n);

vector <int> max = find\_all\_max\_matr(v, n);

cout << endl << "Минимальные элементы: ";

for (int i = 0; i < min.size(); i++) {

cout << min[i] + 1;

if (i < min.size() - 1) cout << ", ";

}

cout << endl;

if (min.size() == 1) cout << "Наименьший элемент: " << min[0] + 1 << endl;

else cout << "Наименьшего элемента нет" << endl;

cout << "Максимальные элементы: ";

for (int i = 0; i < max.size(); i++) {

cout << max[i] + 1;

if (i < max.size() - 1) cout << ", ";

}

cout << endl;

if (max.size() == 1) cout << "Наибольший элемент: " << max[0] + 1 << endl;

else cout << "Наибольшего элемента нет" << endl;

cout << endl;

hasse\_matr(v, n);

}

if (h == 4) {

int n;

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

cin >> n;

int\*\* v;

v = new int\* [n];

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

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

v[i] = new int[n];

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

cin >> v[i][j];

}

}

eq(v, n);

}

if (h == 5) {

int n;

cout << "Введите количество атрибутов и объектов. n = ";

cin >> n;

int\*\* v;

v = new int\* [n];

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

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

v[i] = new int[n];

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

cin >> v[i][j];

}

}

zfg(v, n);

}

cout << endl << endl;

}

return 0;

}

ЗАКЛЮЧЕНИЕ

В ходе данной работы были изучены и реализованы алгоритмы построения основных замыканий, диаграммы Хассе, поиска наибольшего, наименьшего, максимальных, минимальных элементов упорядоченного множества, вычисления системы замыканий, решётки концептов.