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»

студентки 3 курса 331 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Шуликиной Анастасии Александровны

Преподаватель

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

подпись, дата

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ . . . 3
1 Цель работы и порядок её выполнения . . . 4
2 Теория . . . 5
2.1 Системы замыкания на множестве бинарных отношений . . . 5
2.1.1 Алгоритм построения замыкания отношения относительно свойства рефлексивности . . . 6
2.1.2 Алгоритм построения замыкания отношения относительно свойства симметричности . . . 6
2.1.3 Алгоритм построения замыкания отношения относительно свойства транзитивности . . . 6
2.1.4 Алгоритм построения замыкания отношения относительно свойства эквивалентности . . . 7
2.2 Фактор-множества бинарных отношений . . . 7
2.2.1 Алгоритм построения фактор-множества бинарных отношений . . . 7
2.2.2 Алгоритм построения системы представителей фактор-множества бинарных отношений . . . 8
2.3 Отношения порядка и диаграмма Хассе . . . 8
2.3.1 Алгоритм построения диаграммы Хассе . . . 9
2.3.2 Алгоритм поиска минимальных и наименьших элементов упорядоченного множества . . . 10
2.3.3 Алгоритм поиска максимальных и наибольшего элементов упорядоченного множества . . . 10
2.4 Контекст и концепт . . . 11
2.4.1 Алгоритм вычисление системы замыканий . . . 13
2.4.2 Алгоритм вычисления решетки концептов . . . 13
3 Программная реализация рассмотренных алгоритмов . . . 15
3.1 Результаты тестирования программы . . . 15
3.2 Код программы, на основе рассмотренных алгоритмов, на языке C++ . . . 17

ВВЕДЕНИЕ

В данной лабораторной работе поставлена задача рассмотрения отношения эквивалентности, фактор-множества, определения отношения порядка и диаграммы Хассе, определения контекста и концепта, написание алгоритмов для изученных тем.

1 Цель работы и порядок её выполнения

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

Порядок выполнения работы:

1. Разобрать определения отношения эквивалентности, фактор-множества. Разработать алгоритмы построения эквивалентного замыкания бинарного отношения и системы представителей фактор-множества .
2. Разобрать определения отношения порядка и диаграммы Хассе. Разработать алгоритмы вычисления минимальных (максимальных) и наименьших (наибольших) элементов и построения диаграммы Хассе.
3. Разобрать определения контекста и концепта. Разработать алгоритм вычисления решетки концептов.

2 Теория

2.1 Системы замыкания на множестве бинарных отношений

Замыканием отношения R относительно свойства P называется такое множество R^*, что:

1. $R \subset R^*$
2. R^* обладает свойством P.
3. R^* является подмножеством любого другого отношения, содержащего R и обладающего свойством P.

То есть R^* является минимальным надмножеством множества R, выдерживается P.

Множество Z подмножеств множества A называется системой замыканий, если оно замкнуто относительно пересечений, т.е. выполняется $\cap B \in Z$ для любого подмножества $B \subset Z$.

В частности, для $\oslash \subset Z$ выполняется $\cap \oslash = A \in Z$

На множестве всех бинарных отношений между элементами множества A^2 следующие множества являются системами замыканий:

1. Z_r – множество всех рефлексивных бинарных отношений между элементами множества A,
2. Z_s – множество всех симметричных бинарных отношений между элементами множества A,
3. Z_t – множество всех транзитивных бинарных отношений между элементами множества A,
4. $Z_{eq} = Eq(A)$ – множество всех отношений эквивалентности на множестве A.

Множество Z_{as} всех антисимметричных бинарных отношений между элементами множества A не является системой замыкания.

На множестве $P(A^2)$ всех бинарных отношений между элементами множества A следующие отображения являются операторами замыканий:

1. $f_r(\rho) = \rho \cup \triangle_A$ – наименьшее рефлексивное бинарное отношение, содержащее отношение $\rho \subset A^2$,
2. $f_s(\rho) = \rho \cup \rho^{-1}$ – наименьшее симметричное бинарное отношение, содержащее отношение $\rho \subset A^2$,
3. $f_t(\rho) = \cup_{n=1}^{\infty} \rho^n$ – наименьшее транзитивное бинарное отношение, содержащее отношение $\rho \subset A^2$,

4. $f_{eq}(\rho) = f_t f_s f_r(\rho)$ – наименьшее отношение эквивалентности, содержащее отношение $\rho \subset A^2$.

2.1.1 Алгоритм построения замыкания отношения относительно свойства рефлексивности

Вход. Матрица $M(\rho)$ бинарного отношения p размерностью $N \times N$.

Выход. Замыкание относительно свойства рефлексивности.

Шаг 1. Создать копию ρ_1 отношения ρ.

Шаг 2. Пустить цикл по i от 1 до N.

Шаг 2.1. Присваеваем $M_{ii} = 1$, добавляем пару (i, i) в ρ_1.

Шаг 3. return ρ_1.

Трудоемкость алгоритма $O(N)$

2.1.2 Алгоритм построения замыкания отношения относительно свойства симметричности

Вход. Матрица $M(\rho)$ бинарного отношения ρ размерностью $N \times N$.

Выход. Замыкание относительно свойства симметричности.

Шаг 1. Создать копию ρ_1 отношения ρ.

Шаг 2. Пустить цикл по i от 1 до N и цикл по j от 1 до N.

Шаг 2.1. Если $M_{ij} = 1$, то присваеваем $M_{ji} = 1$, добавить пару (j, i) в ρ_1.

Шаг 3. return ρ_1.

Трудоемкость алгоритма $O(N^2)$

2.1.3 Алгоритм построения замыкания отношения относительно свойства транзитивности

Вход. Матрица $M(\rho)$ бинарного отношения ρ размерностью $N \times N$.

Выход. Замыкание относительно свойства симметричности.

Шаг 1. Создать копию ρ_1 отношения ρ.

Шаг 2. Пустить цикл по i от 1 до N, цикл по j от 1 до N, цикл по k от 1 до N и цикл по p от 1 до N.

Шаг 2.1. Проверяем все значения для M_{ij}, если значение равно 1, то для j проверяем все значения M_{jk}, если значение какого-либо k равно 1, то присваеваем $M_{ik} = 1$, добавляем пару (i, k) в ρ_1.

Шаг 3. return ρ_1.

Трудоемкость алгоритма $O(N^4)$

2.1.4 Алгоритм построения замыкания отношения относительно свойства эквивалентности

Вход. Матрица $M(\rho)$ бинарного отношения ρ размерностью $N \times N$.

Выход. Замыкание относительно свойства эквивалентности.

Шаг 1. Создать копию ρ_1 отношения ρ.

Шаг 2. По очереди для ρ_1 вызвать алгоритмы построения замыкания рефлексивности, симметричности и транзитивности.

Шаг 3. return ρ_1.

Трудоемкость алгоритма $O(N^4)$

2.2 Фактор-множества бинарных отношений

Срезы $\rho(a)$ называются классами эквивалентности по отношению ρ и сокращенно обозначаются символом $[a]$.

Пусть A – не пустое множество. Тогда фактор-множеством множества A по отношению эквивалентности ρ называется множество A/ρ всех классов эквивалентности $\{[a] : a \in A\}$.

Подмножество $T \subset A$ называется полной системой представителей классов эквивалентности ε на множестве A если:

1. $\varepsilon(T) = A$;
2. из условия $t_1 \equiv t_2(\varepsilon)$ следует $t_1 = t_2$.

2.2.1 Алгоритм построения фактор-множества бинарных отношений

Вход. Матрица $M(\rho)$ бинарного отношения ρ размерностью $N \times N$.

Выход. Фактор-множество ρ_1.

Шаг 1. Создаём список списков ρ_1 и массив посещенных вершин use. Изначально все вершины не посещены $use = false$.

Шаг 2. Проверяем для всех i от 1 до N на посещение $use[i]$.

Шаг 2.1. Если i не является посёщенным, то помечаем его $use[i] = true$, как посещённый, и добавляем i в новый список x.

Шаг 2.2. Также для всех j от 1 до N проверяем M_{ij}.

Шаг 2.2.1 Также, проверяем, если $M_{ij} = 1$ и $use[j] = false$ - не посещённый, помечаем $use[j] = true$, как посещённый, и добавляем j в x.

Шаг 2.3. По окончании цикла дабавляем в ρ_1 x.

Шаг 3. Выводим фактор-множество ρ_1.

Трудоемкость алгоритма $O(N^2)$

2.2.2 Алгоритм построения системы представителей фактор-множества бинарных отношений

Вход. Фактор-множество ρ.

Выход. Система представителей ρ_1 фактор-множества ρ.

Шаг 1. По возрастанию сортируем все срезы фактор-множества ρ от 1 до K.

Шаг 2. Берём из отсортированного списка срезов наименьший элемент x и добавляем в ρ_1.

Шаг 3. Выводим ρ_1.

Трудоемкость алгоритма $O(N^2)$

2.3 Отношения порядка и диаграмма Хассе

Бинарное отношение ω на множестве A называется отношением порядка (или просто порядком), 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 $\leq\cdot$. вместо $(a, b) \in \omega$ принято писать $a \leq b$ (читается «a меньше или равно b», «a содержится в b» или «b больше или равно a», «b содержит a»).

Запись $a < b$ означает, что $a \leq b$ и $a \neq b$.

Запись $a < \cdot b$ означает, что $a \leq b$ и нет элементов x, удовлетворяющих условию $a < x < b$. В этом случае говорят, что элемент b покрывает элемент a или что элемент a покрывается элементом b.

Элементы, $a, b \in A$ называются сравнимыми, если $a \leq b$ или $b \leq a$, и несравнимыми в противном случае.

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 $(\forall x \in A) x \leq a \Longrightarrow x = a$,

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

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

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

Очевидно, что наименьший (соответственно, наибольший) элемент является минимальным (соответственно, максимальным), но в общем случае обратное неверно.

Для любого конечного упорядоченного множества $A = (A, \leq)$ справедливы следующие утверждения:

1. любой элемент множества A содержится в некотором максимальном элементе и содержит некоторый минимальный элемент;
2. если упорядоченное множество A имеет один максимальный (соответственно, минимальный) элемент, то он является наибольшим (соответственно, наименьшим) элементом этого множества.

Бинарное отношение ω на множестве A 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 $\leq$: вместо $(a, b) \in \omega$ принято писать $a \leq b$.

Диаграмма Хассе – вид диаграмм, используемый для представления конечного частично упорядоченного множества в виде рисунка его транзитивного сокращения. Диаграмма представляет элементы множества A точками плоскости и пары $a < \cdot b$ представляет линиями, идущими вверх от элемента a к элементу b.

Для любого конечного упорядоченного множества $A = (A, \leq)$ справедливы следующие утверждения:

1. любой элемент множества A содержится в некотором максимальном элементе и содержит некоторый минимальный элемент;
2. если упорядоченное множество A имеет один максимальный (соответственно, минимальный) элемент, то он является наибольшим (соответственно, наименьшим) элементом этого множества.

2.3.1 Алгоритм построения диаграммы Хассе

Вход. Конечное упорядоченное множество $A = (A, \leq)$, обозначим как ρ.

Выход. Диаграмма Хассе res.

<u>Шаг 1.</u> Создаём переменную $lvl = 0$ и список списков res.

Шаг 2. Поочерёдно находим в ρ все минимальные элементы, добавляем их в res, а из ρ удаляем эти элементы, $lvl = lvl + 1$.

Шаг 2.2. Выполняем пока $\rho \neq 0$.

Шаг 3. Создаём список пар $pairs$ для всех i от $res.size$ до 0.

Шаг 3.1. Для всех j от 0 до $res[i].size$.

Шаг 3.2. Для всех k от 0 до $res[i-1].size$.

Шаг 3.3. Если $res[i][j] \leq res[i+1][k]$, добавляем пару $(res[i][j], res[i+1][k])$ в $pairs$.

Шаг 4. Выводим res и $pairs$.

Трудоемкость алгоритма $O(N^4)$

2.3.2 Алгоритм поиска минимальных и наименьших элементов упорядоченного множества

Вход. Конечное упорядоченное множество $A = (A, \leq)$, обозначим как ρ.

Выход. Список минимальных элементов res и наименьший элемент min.

Шаг 1. Создаём переменную min и список res.

Шаг 2. Для $\forall i \in A$, i от 1 до N заводим переменную $chek = 0$.

Шаг 2.1. $\forall j \in A$, j от 1 до N если $\rho[j] \leq \rho[i]$ && $i \neq j$, то $chek = chek{+}1$.

Шаг 2.2. Если $chek = 0$, то i кладём в res.

Шаг 3. Если в res больше одного элемента, то наименьшего элемента нет. Иначе наименшим является единственный элемент из res.

Шаг 4. Выводим res и min, если есть.

Трудоемкость алгоритма $O(N^2)$

2.3.3 Алгоритм поиска максимальных и наибольшего элементов упорядоченного множества

Вход. Конечное упорядоченное множество $A = (A, \leq)$, обозначим как ρ.

Выход. Список максимальных элементов res и наибольший элемент max.

Шаг 1. Создаём переменную max и список res.

Шаг 2. Для $\forall i \in A$, i от 1 до N заводим переменную $chek = 0$.

Шаг 2.1. $\forall j \in A$, j от 1 до N если $\rho[i] \leq \rho[j]$ && $i \neq j$, то $chek = chek{+}1$.

Шаг 2.2. Если $chek = 0$, то i кладём в res.

Шаг 3. Если в res больше одного элемента, то наибольшего элемента нет. Иначе наибольшим является единственный элемент из res.

Шаг 4. Выводим res и max, если есть.

Трудоемкость алгоритма $O(N^2)$

2.4 Контекст и концепт

Теорема Галуа. Пусть $\rho = A \times B$ – произвольное бинарное отношение между элементами множеств A и B. Для любого подмножества $X \subset A$ определим подмножество $X' \subset B$ по формуле

$$X' = \{b \in B | (x, b) \in \rho \forall x \in X\}$$

и для любого подмножества $Y \subset B$ определим подмножество $Y' \subset A$ по формуле

$$Y' = \{a \in A | (a, y) \in \rho \forall y \in Y\}.$$

Тогда отображение $X \to X'$ и $Y \to Y'$ множеств $P(A)$ и $P(B)$ друг в друга обладают следующими свойствами:

1. $X_1 \subset X_2 \Rightarrow X_2' \subset X_1'$;
2. $Y_1 \subset Y_2 \Rightarrow Y_2' \subset Y_1'$;
3. $X \subset X'', Y \subset Y''$;
4. $X''' \subset X', Y''' \subset Y'$.

Пара таких отображений называется соответствием Галуа, которое определяет два оператора замыкания f_A и f_B на множествах A и B по формулам:

$$f_A(X) = X'' \forall X \subset A,$$

$$f_B(Y) = Y'' \forall Y \subset B.$$

При этом отображения $X \to X'$ и $Y \to Y'$ определяют антиизоморфизмы между системами замыканий Z_{f_A} и Z_{f_B} на множествах A и B, т.е. это биекции, которые удовлетворяют условиям:

$$X \subset Y \Leftrightarrow X' \supset Y' \forall X, Y \in Z_{f_A},$$

$$X \subset Y \Leftrightarrow X' \supset Y' \forall X, Y \in Z_{f_B},$$

Бинарное отношение $\rho \subset G \times M$ между элементами множеств G и M можно рассматривать как базу данных с множеством объектов G и множеством атрибутов M. Такая система называется также контекстом и определяется следующим образом.

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.

Контекст $K = (G, M, \rho)$, наглядно изображается таблицей, в которой строки помечены элементами множества G, столбцы помечены элементами множества M и на пересечении строки с меткой $g \in G$ и столбца с меткой $m \in M$ стоит элемент $k_{g,m} = \begin{cases} 1, if(g,m) \in \rho, \\ 0, if(g,m) \in \rho. \end{cases}$

Контекст $K = (G, M, \rho)$ с множеством объектов $G = 1, 2, 3, 4$, множеством атрибутов $M = a, b, c, d$ и отношением

$$\rho = (1, a), (1, c), (2, a), (2, b), (3, b), (3, d), (4, b), (4, d)$$

изображается таблицей.

Таблица 1.

K	a	b	c	d
1	1	0	1	0
2	1	1	0	0
3	0	1	0	1
4	0	1	0	1

По теореме Галуа для контекста $K = (G, M, \rho)$ отображения

$$\varphi : P(G) \mapsto P(M), \psi : P(M) \mapsto P(G),$$

которые для $X \subset G$, $Y \subset M$ определяются по формулам:

$$\varphi(X) = X' = m \in M|(x, m) \in \rho \forall x \in X, \psi(Y) = Y' = g \in G|(g, y) \in \rho \forall y \in Y,$$

образуют соответствие Галуа между подмножествами множеств G и M, которое определяет на этих множествах операторы замыкания по формулам:

$$f_G(X) = X'' = \psi(\varphi(X)), f_M(Y) = Y'' = \varphi(\psi(Y)),$$

где $X \subset G$, $Y \subset M$.

При этом отображение φ и ψ определяют антиизоморфизмы между системами замыканий Z_{f_G} и Z_{f_M} на множестве G и M, т.е. это биекции, которые удовлетворяют условиям:

$$X \subset Y \Leftrightarrow \varphi(X) \supset \varphi(Y) for X, Y \in Z_{f_G},$$

$$X \subset Y \Leftrightarrow \psi(X) \supset \psi(Y) for X, Y \in Z_{f_M}.$$

Упорядоченная пара (X, Y), замкнутых множеств $X \in Z_{f_G}$, $Y \in Z_{f_M}$, удовлетворяющих условиям $\varphi(X) = Y$, $\psi(Y) = X$, назывется концептом контекста $K = (G, P, \rho)$. При этом компонента X называется объёмом 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.

2.4.1 Алгоритм вычисление системы замыканий

Вход. Контекст $K = (G, P, \rho)$.

Выход. Система замыканий Z_{f_G}.

Шаг 1. Создаём список списков res и кладем туда G.

Шаг 2. Для $\forall i \in P$ заводим список $count$.

Шаг 2.1. $\forall i \in P$, если $\rho[j][i] = 1$, то кладем j в $count$.

Шаг 2.2. Если в списке res еще нет такого $count$, то кладём в res $count$.

Шаг 3. Пересекаем $count$ со всеми списками из res.

Шаг 3.2. Если пересечения еще нет в списке res, то кладём его в res.

Трудоемкость алгоритма $O(N^4)$

2.4.2 Алгоритм вычисления решетки концептов

Вход. Контекст $K = (G, P, \rho)$.

Выход. Решетка концептов $C(K)$.

Шаг 1. Заводим список списков пар res. Вычисляем систему замыканий Z_{f_G} по алгоритму 2.5.1.

Шаг 2. Для $\forall$ подмножеств $i \subset Z_{f_G}$ рассматриваем $\varphi(j_i)$ $\forall j_i \in i$.

Шаг 2.1. Пересекаем все $\varphi(j_i)$ $\forall j_i \in i$ и ставим это пересечение в пару с i.

Шаг 2.2. Добавляем эту пару в res.

Шаг 3.2. Выводим res.

Трудоемкость алгоритма $O(N^5)$

3 Программная реализация рассмотренных алгоритмов

3.1 Результаты тестирования программы

На рисунках 1-5 можно увидеть работу, реализуемой программы, по рассмотренным алгоритмам.

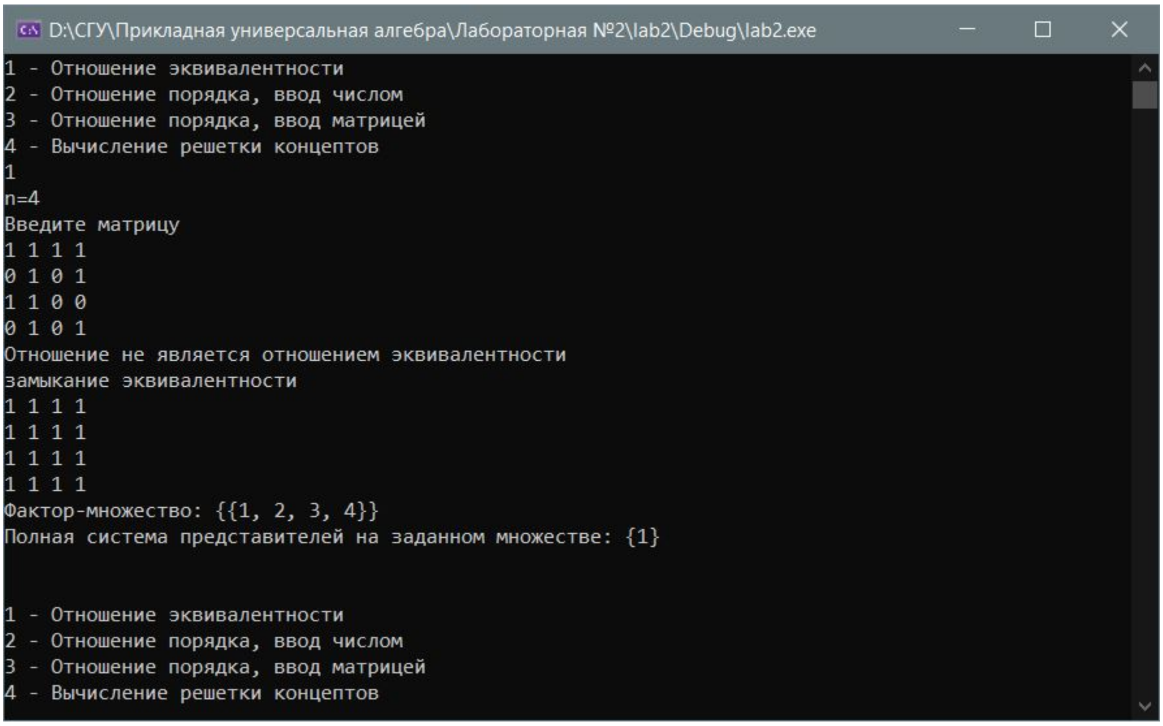

Рисунок 1 – Тест программы для отношения эквивалентности

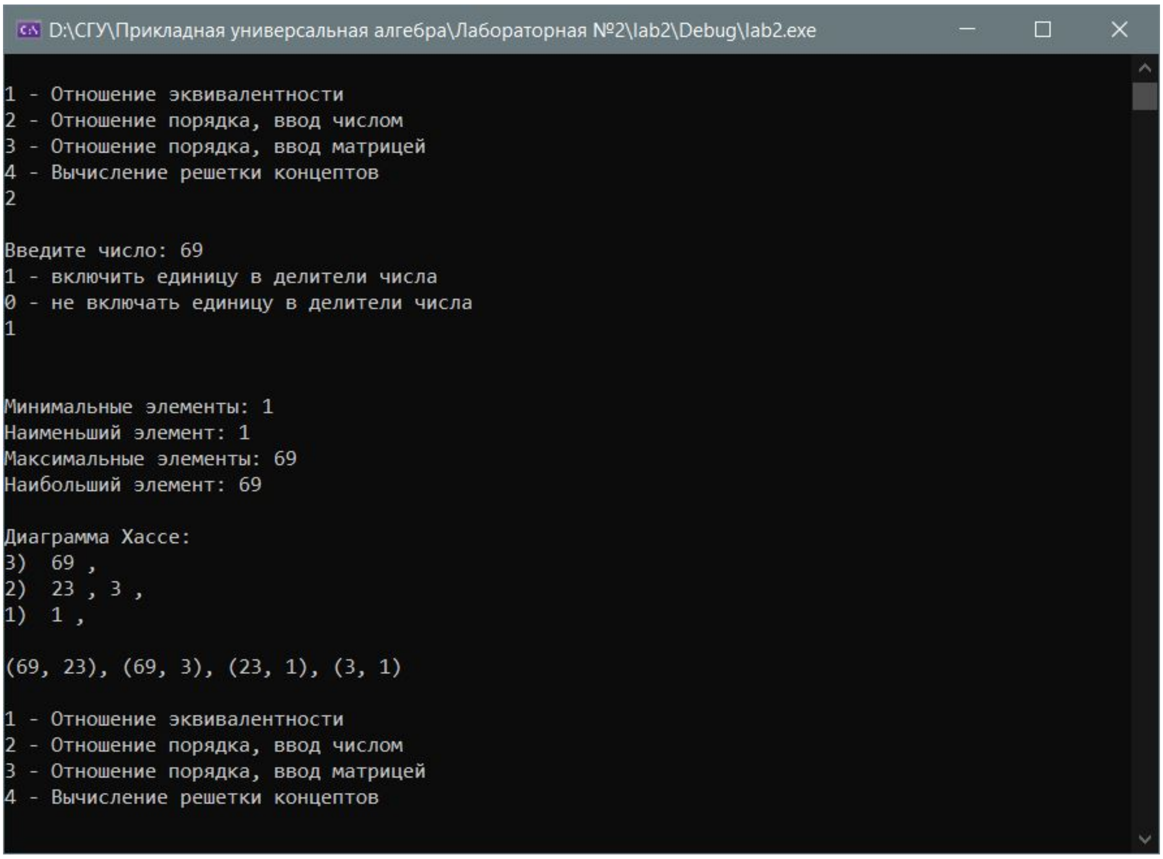

Рисунок 2 – Тест программы для отношения порядка числа(включая 1)

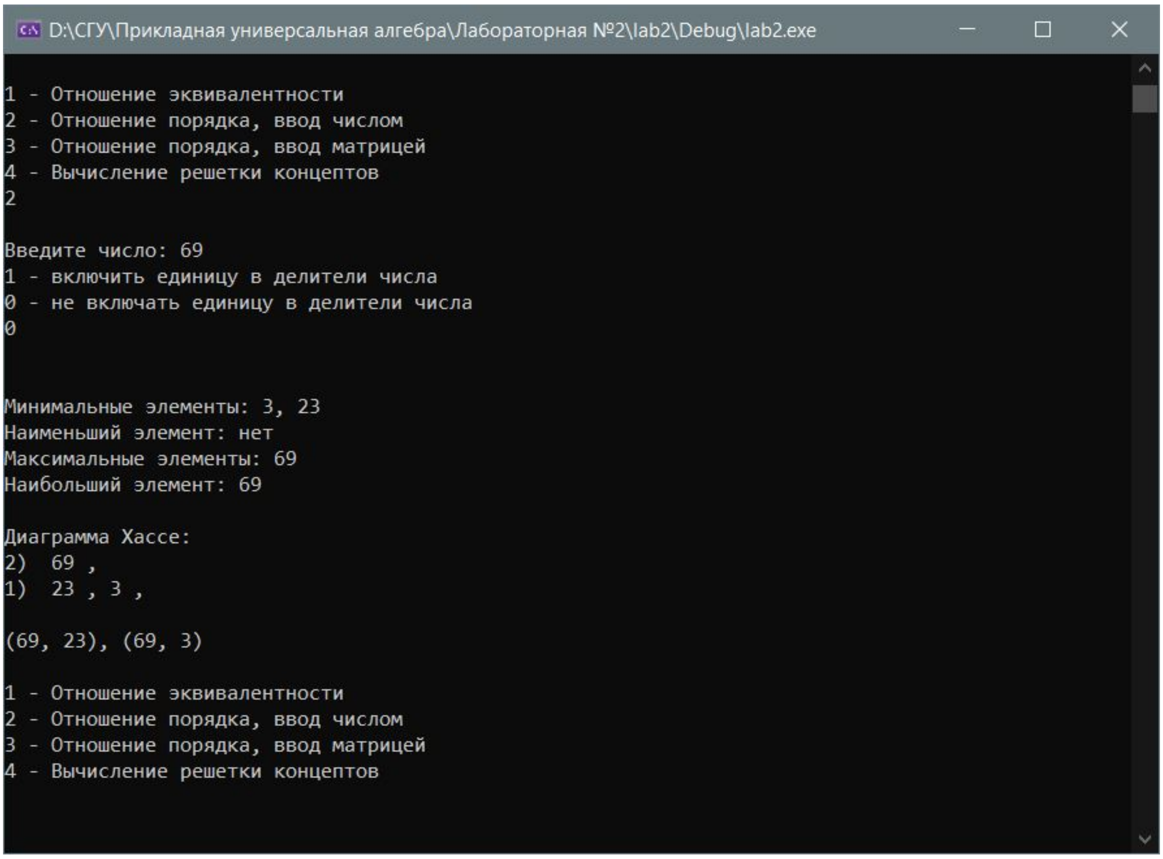

Рисунок 3 – Тест программы для отношения порядка числа(не включая 1)

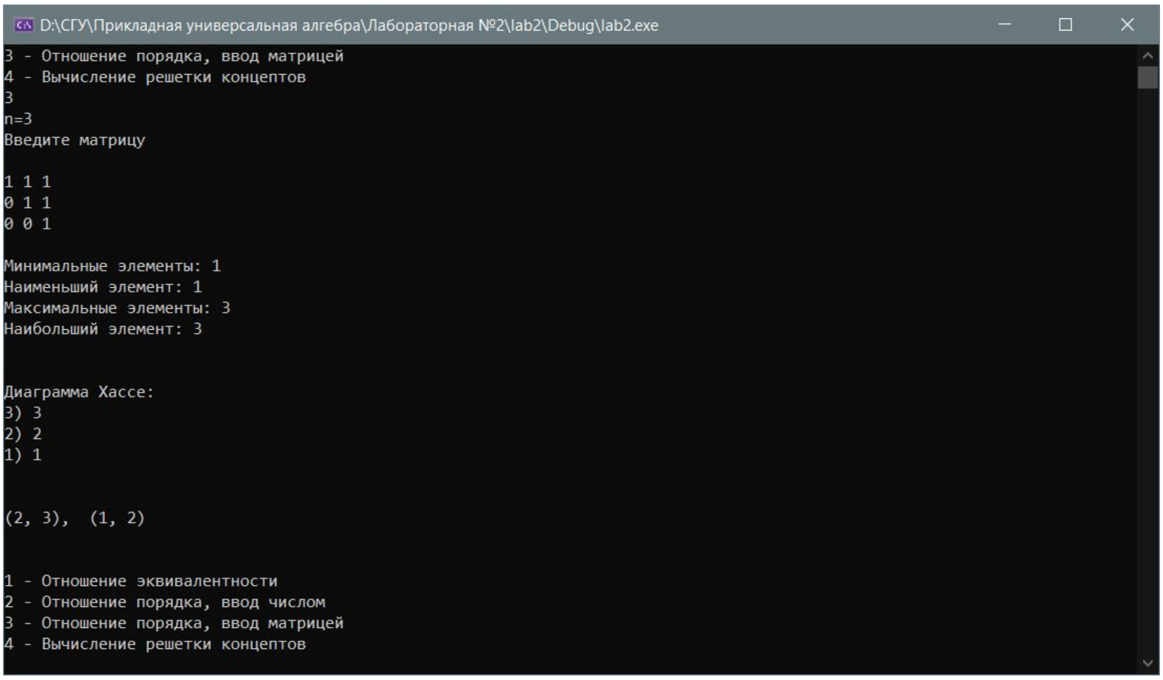

Рисунок 4 – Тест программы отношения порядка матрицы

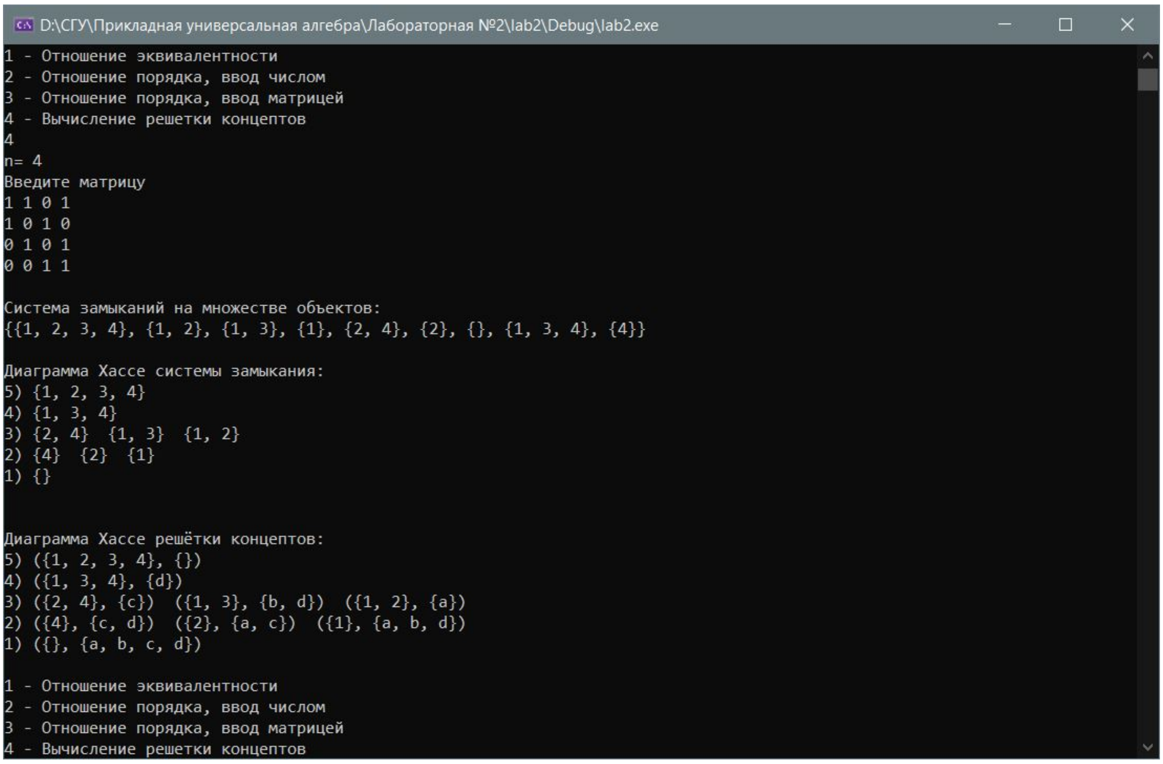

Рисунок 5 – Тест программы вычисления решетки концептов

3.2 Код программы, на основе рассмотренных алгоритмов, на языке C++

```
#include <iostream>
#include <vector>
```

```
using namespace std;

int symmetry1 = 0, reflexivity1 = 0,
transitivity1 = 0, antisymmetry1 = 0;

void symmetry(int** a, int n)
{
        int not_symmetry1 = 0;
        for (int i = 0; i < n; i++)
        {
                for (int j = 0; j < n; j++)
                {
                        if (a[i][j] != a[j][i])
                        not_symmetry1++;

                }
        }
        if (not_symmetry1 == 0)
        cout << "relation is symmetrical" << endl;
        symmetry1++;
}

void antisymmetry(int** a, int n)
{
        int not_antisymmetry1 = 0;
        for (int i = 0; i < n; i++)
        {
                for (int j = 0; j < n; j++)
                {
                if (a[i][j] == 1
                && a[j][i] == 1
                && i != j)
                not_antisymmetry1++;
```

```
				}
			}
			if (not_antisymmetry1 == 0) {
			cout << "relation is
			antisymmetrical" << endl;
			antisymmetry1++;
			}
}

void reflexivity(int** a, int n)
{
			int chek = 0;
			for (int i = 0; i < n; i++)
			{
				if (a[i][i] == 1)
				chek++;
			}
			if (chek == n) {
				cout << "relation is reflexivity" << endl;
				reflexivity1++;
			}
}

void transitivity(int** a, int n)
{
			int chek = 0;
			int not_transitivity1 = 0;
			for (int i = 0; i < n; i++)
			{
				for (int j = 0; j < n; j++)
				{
				if (a[i][j])
				{
				for (int k = 0; k < n; k++)
```

```
                    {
                    if (a[j][k] && !a[i][k])
                    {
                    not_transitivity1++;
                    }
                    else
                    {
                    chek++;
                    }
                    }
                    }
                    }
          }
          if (chek == 1) {
          cout << "relation is transitivity" << endl;
          transitivity1++;
          }
}

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

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

void pr_transitivity(int** a, int n)
```

```
{
        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 p = 0; p < n; p++)     {
                if (a[k][p] == 1)
                a[j][p] = 1;
                }
                }
                }
                }
        }
}

void pr(int** a, int n) {
        int** a1;
        a1 = new int* [n];
        for (int i = 0; i < n; i++)
        {
                a1[i] = new int[n];
                for (int j = 0; j < n; j++)
                {
                        a1[i][j] = a[i][j];
                }
        }

        pr_symmetry(a1, n);
        pr_reflexivity(a1, n);
        pr_transitivity(a1, n);
        cout << "equivalence closure" << endl;

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

```
                cout << a1[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 (a1[i][j] && count[j]) {
                                count[j] = 0;
                                srez.push_back(j);
                                }
                        }
                        count[i] = 0;
                        res.push_back(srez);
                }
        }

        cout << "factor-set: {";
                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 << "Full system of representatives: {";
        for (int i = 0; i < res.size(); i++) {
            cout << res[i][0] + 1;
            if (i != res.size() - 1)
            cout << ", ";
        }
        cout << "}" << endl << endl;

}

void rel_equivalence() {
    int n;
    cout << "n=";
    cin >> n;
    int** a;
    a = new int* [n];
    cout << "Enter matrix\n";
    for (int i = 0; i < n; i++)
    {
        a[i] = new int[n];
        for (int j = 0; j < n; j++)
        {
            cin >> a[i][j];
        }
    }

    symmetry(a, n);
    reflexivity(a, n);
    transitivity(a, n);
    antisymmetry(a, n);
```

```
        if (symmetry1 > 0
        && reflexivity1 > 0
        && transitivity1 > 0)
        cout << The relation is
        an equivalence relation\n";
        else {
                cout << "The relation isn't
                an equivalence relation\n";
                pr(a, n);
        }
}

vector <pair <int, int> > all_min (vector <int> a) {
        vector <pair<int, int> > res;
        res.push_back(make_pair(a[0], 0));
        for (int i = 1; i < a.size(); i++) {
                int checker = 0;
                for (int j = 0; j < res.size(); j++)
                if (a[i] % res[j].first == 0)
                checker++;
                if (checker == 0)
                res.push_back(make_pair(a[i], i));
        }
        return res;
}

void hasse(vector <int> a) {
        vector <vector <int>> res;
        int lvl = 0;
        cout << endl << "Hasse diagram: \n";
        while (a.size() > 0) {
                vector <pair<int, int>> A_i = all_min(a);
                vector <int> help;
        for (int i = A_i.size() - 1; i >= 0; i--) {
```

```
            a.erase(a.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--) {
                    cout << i + 1 << ") ";
                    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 n, int k) {
vector <int> res;
for (int i = 2 - k; i < n / 2.0 + 1; i++) {
        if (n % i == 0)
        res.push_back(i);
}
res.push_back(n);
return res;
}

void order_number() {
int n;
cout << "\nEnter the number: ";
cin >> n;

int one, k;
cout << "1 - with 1 \n0 - without 1 \n";
cin >> one;
if (one)
k = 1;
else
k = 0;
cout << endl;
vector <int> res = find_all(n, k);
vector <pair<int, int>> min = all_min(res);
cout << endl << "Minimum elements: ";
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 << "smallest element: "
<< min[0].first << endl;
else cout << "smallest element: no" << endl;
```

```
cout << "Maximum elements: " << n << endl;
cout << "largest element: " << n << endl;

hasse(res);
}

vector <int> 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> 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);
cout << endl << "Hasse diagram: \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 = 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--) {
        cout << i + 1 << ") ";
        for (int j = 0; j < res[i].size(); j++) {
                cout << res[i][j] + 1 << " ";
        }
        cout << endl;
}
cout << endl << "\n";
hasse_pair(v_real, n, res);
}

void order_matrix() {
int n;
cout << "n=";
cin >> n;
```

```
vector <vector <int>> v;
v.resize(n);
cout << "Enter matrix \n" << 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);
        }
}

if (antisymmetry1 > 0
&& reflexivity1 > 0
&& transitivity1 > 0)
cout << "The relation is an order relation\n";
else {
                cout << "The relation
                isn't an order relation\n";
                return;
        }

vector <int> min = all_min_matr(v, n);
vector <int> max = all_max_matr(v, n);

cout << endl << "Minimum elements: ";
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 << "smallest element: " << min[0] + 1 << endl;
else cout << "smallest element: no" << endl;
```

```
cout << "Maximum elements: ";
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 << "largest element: " << max[0] + 1 << endl;
else cout << "largest element: no" << endl;
cout << endl;
hasse_matr(v, n);
}

vector<int> p(vector<int> a1, vector<int> a2) {
vector<int> res;
for (int i = 0; i < a1.size(); i++)
for (int j = 0; j < a2.size(); j++)
if (a1[i] == a2[j])
res.push_back(a1[i]);
return res;
}

bool check(vector<vector<int>> a1, vector<int> a2) {
for (int i = 0; i < a1.size(); i++) {
        if (a1[i].size() == a2.size()) {
                int a = a2.size();
                for (int j = 0; j < a1[i].size(); j++) {
                        if (a1[i][j] == a2[j])
                        a--;
                }
                if (a == 0)
                return false;
        }
}
```

```
return true;
}

vector <pair<vector <int>, int>>
all_min_zm(vector<vector<int>> a) {
vector <pair<vector <int>, int>> res;
for (int i = 0; i < a.size(); i++) {
        int count = 0;
        for (int j = 0; j < a.size(); j++) {
                if (p(a[i], a[j]) ==
                a[i] || p(a[i], a[j]) != a[j]) {
                        count++;
                }
        }
        if (count == a.size())
        res.push_back(make_pair(a[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>>> a, 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(a.size());
```

```
for (int i = 0; i < a.size(); i++) {
        for (int j = 0; j < a[i].size(); j++) {
                vector<int> count;
                int checker = 0;
                for (int k = 0; k < a[i][j].size(); k++) {
                        checker++;
                        if (k == 0)
                        count = fi(matr, a[i][j][k], n);
                        else
                count = p(count, fi(matr, a[i][j][k], n));
                }
                if (checker > 0)
                res[i].push_back(make_pair(a[i][j], count));
                else {
                vector<int> help;
                for (int i = 0; i < n; i++)
                help.push_back(i);
                res[i].push_back(make_pair(a[i][j],
                help));
                }
        }
}
cout << endl << "Hasse diagram of concept lattice:" << endl;
for (int i = res.size() - 1; i >= 0; i--) {
        cout << i + 1 << ") ";
        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_zm(vector<vector<int>> a, int** matr, int n) {
vector<vector<vector<int>>> res;
int lvl = 0;
while (a.size() > 0) {
        vector <pair<vector <int >, int>> A_i = all_min_zm(a);
        vector<vector<int>> help;
        for (int i = A_i.size() - 1; i >= 0; i--) {
                a.erase(a.begin() + A_i[i].second);
                help.push_back(A_i[i].first);
        }
        res.push_back(help);
        lvl++;
}
cout << "Hasse diagram of the closing system: " << endl;
for (int i = res.size() - 1; i >= 0; i--) {
        cout << i + 1 << ") ";
        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;
c(res, matr, n);
}

vector<vector<int>> zm (int** a, 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 (a[j][i])
                count.push_back(j);
        }
```

```
        if (check(res, count))
        res.push_back(count);
        for (int i = 0; i < res.size(); i++) {
                vector<int> new_count = p(res[i], count);
                if (check(res, new_count))
                res.push_back(new_count);
        }
}

cout << endl << "Closure system on a set
                of objects:" << 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_zm(res, a, n);

return res;
}

void consept() {
int n;
cout << "n= ";
cin >> n;
```

```
int** a;
a = new int* [n];
cout << "Enter matrix \n";
for (int i = 0; i < n; i++) {
        a[i] = new int[n];
        for (int j = 0; j < n; j++) {
                cin >> a[i][j];
        }
}
zm(a, n);
}

int main()
{
setlocale(LC_ALL, "ru");
for (;;) {
        cout << "1 - Equivalence relation
        \n2 - Order relation, input as a number
        \n3 - Order relation, input by matrix
        \n4 - Concept Lattice Computation \n";
        int x;
        cin >> x;
        switch (x) {
                case 1:
                rel_equivalence();
                cout << endl;
                break;
                case 2:
                order_number();
                cout << endl;
                break;
                case 3:
                order_matrix();
                cout << endl;
```

```
                    break;
                    case 4:
                    consept();
                    cout << endl;
                    break;
          }
}
}
}
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были рассмотрены и реализованы основные свойства бинарных отношений и операции замыкания бинарных отношений, построена диаграмма Хассе, реализованы алгоритмы поиска минимального, максимального, наименьшего, наибольшего элементов упорядоченного множества, вычисления системы замыканий и решётки концептов.