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

Зиминой Ирины Олеговны

Преподаватель

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

подпись, дата

Саратов 2022

СОДЕРЖАНИЕ

1. Цель работы и порядок 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. Определение отношения эквивалентности и эквивалентного замыкания бинарного отношения

Отношение является **отношением эквивалентности**, если бинарное отношение рефлексивное, симметричное и транзитивное.

Бинарное отношение $\rho \subset A \times B$ называется:

– рефлексивным, если $(a, a) \in \rho$ для любого $a \in A$;
– симметричным, если $(a, b) \in \rho \implies (b, a) \in \rho$, для любых $a, b \in A$;
– транзитивным, если $(a, b) \in \rho$ и $(b, c) \in \rho \Rightarrow (a, c) \in \rho$ для любых $a, b, c \in A$.

Для обозначения эквивалентности ε используется инфиксная запись с помощью символа $\equiv$.

Замыканием отношения R относительно свойства P называют 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.

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

1) $f_r(\rho) = \rho \cup \Delta_A$ – 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) $f_t(\rho) = \bigcup_{n=1}^{\infty} \rho^n$ – наименьшее транзитивное бинарное отношение, содержащее отношение $\rho \subset A^2$;
4) $f_{e\rho}(\rho) = f_t f_s f_r(\rho)$ – наименьшее отношение эквивалентности, содержащее отношение $\rho \subset A^2$.

Таким образом можно понять, что эквивалентное замыкание строится посредством совокупности замыканий относительно рефлексивности, симметричности и транзитивности.

3. Алгоритм построения эквивалентного замыкания бинарного отношения

3.1 Построение исходного замыкания.

Для построения любых замыканий, сначала нужно вывести исходное отношение.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: исходное отношение.

Описание алгоритма:

1) Запускается цикл прохода по матрице от элемента $M[0][0]$ до элемента $M[N-1][N-1]$.
2) Если $M[i][j] = 1$, то пара (i, j) добавляется в вектор пар $a(N^2)$.
3) После выхода из цикла, выводится построенный вектор пар.

3.2 Построение эквивалентного замыкания.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: замыкание относительно свойства эквивалентности.

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N-1$.
2) Если $M[i][i] = 0$, то пара (i, j) добавляется в вектор пар a.
3) В вектор пар a, добавляются эти же пары, только в формате (j, i).
4) Строится новая двоичная матрица W, с размерностью $N \times N$, которая является квадратом исходной матрицы бинарного отношения p.
5) Создается матрица $copy\_M$, с размерностью $N \times N$, которая является копией исходной матрицы.

6) Пока бинарное отношение представленное матрицей $copy\_M$ не транзитивное.

7) Проводится сравнение матриц $copy\_M(N * N)$ и $W(N * N)$. Для этого запускается цикл для прохода по двум одновременно: по матрице $copy\_M$ от элемента $copy\_M[0][0]$ до элемента $copy\_M[N-1][N-1]$, по матрице W от элемента $W[0][0]$ до элемента $W[N-1][N-1]$.

8) Если $copy\_M[i][j] = 0$ и $W[i][j] = 1$, то пара (i, j) добавляется в вектор пар a.

9) Элемент $copy\_M[i][j]$ становится равным единице.

10) Если бинарное отношение представленное матрицей $copy\_M$ не транзитивное, то матрица W становится матрицей произведения матриц $copy\_M$ и M.

11) После выхода из цикла (т.е, бинарное отношение представленное матрицей $copy\_M$– транзитивное), выводится построенный вектор пар.

12) С помощью сета убираются дубликаты пар.

13) Выводится построенный сет.

Сложность алгоритма: $O(N^4)$.

4. Определение фактор-множества

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.

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 $t_1 = t_2$.

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

5. Алгоритм построения системы представителей фактор-множества.

Идея алгоритма: для того чтобы вывести представителей фактор-множество, сначала нужно составить фактор-множество. Фактор-множество – это перечисление компонентов сильной связанности графа. Определение**:** две вершины ориентированного графа связаны сильно, если существует путь из одной в другую и наоборот. Иными словами, они обе лежат в каком-то цикле. После нахождения компонентов сильной связанности, выбирается по элементу из каждого компонента.

Входные данные: матрица $M(\rho)$ бинарного отношения ρ с размерностью $N \times N$.

Выходные данные: система представителей фактор-множества.

Описание алгоритма:

Для реализации алгоритма дважды используется обход графа в глубину.

Первый обход в глубину:

1) Функция проходит по матрице M бинарного отношения ρ с размерностью $N \times N$.
2) Помечает пройденную вершину (в матрице представленной единицей), добавлением в вектор.
3) Запускается цикл прохода по индексу i от 0-го до величины размерности заданного графа.
4) При каждом следующем нахождение новой, не помеченной вершины функция вызывает сама себя.
5) После выхода из цикла, в результирующий вектор добавляется обработанная вершина.

Второй обход в глубину:

1) Функция проходит по транспонированной матрице M бинарного отношения ρ с размерностью $N \times N$.
2) Помечает пройденную вершину (в матрице представленной единицей), добавлением в вектор.
3) Добавляет в результирующий вектор вершину.
4) При каждом следующем нахождение новой, не помеченной вершины функция вызывает сама себя.

Основная часть:

1) Вводится матрица M бинарного отношения ρ с размерностью N×N.
2) Одновременно вводится транспонированная матрица M бинарного отношения ρ с размерностью N×N.
3) Запускается цикл прохода по индексу i от 0-го до $N - 1$.
4) Для каждой непомеченной вершины запускается первая функция обхода в глубину.
5) Запускается цикл прохода по индексу i от $N - 1$-го до 0.
6) Для каждой непомеченной вершины запускается вторая функция обхода в глубину.
7) После завершения второй функции, вершина добавляется в вектор результата.
8) Итоговый вектор результата сортируется от большого к меньшему. И выводится.

Сложность алгоритма: $O(N^2)$.

6. Определение отношения порядка

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 $\leq$: 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»).

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

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

- **минимальным**, если $(\forall x \in A)\ x \leq a \Longrightarrow x = a$,
- **максимальными**, если $(\forall x \in A)\ a \leq x \Longrightarrow x = a$,
- **наименьшим**, если $(\forall x \in A)\ a \leq x$,
- **наибольшим**, если $(\forall x \in A)\ x \leq a$.

Наименьший и наибольший элементы упорядоченного множества $(A, \leq)$ принято обозначать 0 и 1.

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

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

1) любой минимальный 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.

7. Алгоритм вычисления минимальных (максимальных) и наименьших (наибольших) элементов

Входные данные: число x.

Выходные данные: множество делителей числа x, наименьший и наибольший элементы, минимальные и максимальные элементы.

Описание алгоритма:

1) Формируется $divisor\_set$ – сет со множеством делителями числа x:

Запускается цикл прохода от $i = 1$ до $i = x$. Если при делении x на i остаток равен нулю, то число i добавляется в сет $divisor\_set$.

2) Выводится сет $divisor\_set$.
3) Выводится наименьший элемент – первый элемент сета $divisor\_set$ и наибольший элемент – последний элемент сета $divisor\_set$.
4) Из сета $divisor\_set$ удаляются первый и последний элементы сета.
5) Формируется $simple\_divisors$ – вектор простых делителей числа x:

Вводятся две переменных: $q = x, w = 2$. Запускается цикл пока $q \neq 1$, внутри цикла проверяется условие, что при делении q на w остаток от деления равен нулю, тогда $q = q \div w$, иначе значение w увеличивается на единицу.

6) Если вектор $simple\_divisors$ не является пустым, то создается $minimal\_elements$ – сет, куда помещаются значения вектора $simple\_divisors$. Выводится сет $minimal\_elements$, что является перечислением минимальных элементов. Иначе выводится сообщение, что минимальный элемент равен единице.
7) Если вектор $simple\_divisors$ не является пустым, то создается $maximum\_elements$ – сет, для хранения максимальных элементов. Запускается цикл прохода от $i = 1$ до $i =$ размерности вектора $simple\_divisors$ и результат деления числа x на $simple\_divisors[i]$ добавляется в $maximum\_elements$. Выводится сет $maximum\_elements$, что является перечислением максимальных элементов. Иначе выводится сообщение, что максимальный элемент – число x.

Сложность алгоритма: $O(x + \log x^2)$.

8. Диаграмма Хассе

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

(Запись $a \lessdot b$ 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).

9. Теоретическое описание построения диаграммы Хассе

Алгоритм построения диаграмма 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 $A \backslash A_1$ 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 $A \backslash (A_1 \cup A_2)$ найти множество A_3 всех минимальных элементов и расположить их в один горизонтальный ряд над вторым уровнем диаграммы (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.

10. Алгоритмы построения диаграммы Хассе

10.1 Диаграмма Хассе для числа.

Входные данные: число x.

Выходные данные: диаграмма Хассе для числа x.

Описание алгоритма:

1) Создается контейнер $connect$, который состоит из числа и сета чисел.

2) С помощью обхода в глубину заполняется контейнер: число – число-делитель числа x, в сете – числа, с которыми связано число-делитель.
3) Создается вектор пар res, где первое число – число-делитель, второе число – уровень, на котором число-делитель находится в диаграмме Хассе.
4) Вектор пар res наполняется элементами контейнера $levels$ (хранящий в себе уровни связи чисел, и вычисляющего при обходе в глубину).
5) Выводится диаграмма Хассе для числа x по условию, что 0 уровень – само число x, а последний уровень – единица, не имеет связей, потому что связь показывается от меньшего уровня к большему.

Сложность алгоритма: $O(x + \log x^3)$.

10.2 Диаграмма Хассе для матрицы.

Входные данные: матрица $matrix$ с размерностью $N \times N$.

Выходные данные: диаграмма Хассе для матрицы $matrix$.

Описание алгоритма:

1) Создается матрица $original\_matrix$ – идентичная введенной матрице $matrix$.
2) Создается сет $elements$, куда помещаются числа от 0 до $N - 1$.
3) Создается вектор векторов чисел $hasse\_levels$.
4) Запускается цикл пока в сете $elements$ есть элементы:

Создается вектор чисел $minimal$ и заполняется минимальными элементами в матрице. В $hasse\_levels$ добавляется $minimal$. Минимальный элемент удаляется из матрицы. Строка(и) матрицы по номеру равная(ые) числу(ам) в $minimal$ заполняются нулями.

5) Создается вектор пар $hasse\_level\_connections$. В котором будут храниться связи созданные на основе $original\_matrix$ и $hasse\_levels$.
6) Выводится диаграмма Хассе по уровням.
7) Выводится список связей в диаграмме Хассе по уровням.

Сложность алгоритма: $O(N^3)$.

11. Определение контекста и концепта

Контекст – это алгебраическая система $K = (G, M, \rho)$, состоящая из множества объектов G, 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 $k_{g,m} = \begin{cases} 1, \text{если } (g, m) \in \rho, \\ 0, \text{если } (g, m) \notin \rho. \end{cases}$

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).

12. Алгоритм вычисления решетки концептов

Входные данные: число N – количество объектов и количество атрибутов, матрица $matrix$ размерностью $N * N$.

Выходные данные: решётка концептов.

Описание алгоритма:

1) Создается вектор сетов A, размерностью N. Куда добавляется номер элемента, который равен единице в матрице $matrix$.
2) Создается сет сетов $inter$, куда помещается вектор сетов A.
3) Создается вектор $full$ размерностью N и последовательно заполняется нулями.
4) В $inter$ добавляется вектор $full$.
5) Создается вектор пар векторов ans. Проходясь по каждому элементу сета $inter$ в конце ans добавляется пустая пара, внутри цикла от $i = 0$ до $i = N - 1$ пустая пара изменяется на значения решетки концептов.
6) Вектор ans сортируется по количеству элементов во втором элементе пары.

7) Выводится полученная решетка концептов от первого до четвертого уровня.

Сложность алгоритма: $O(N^3)$.

13. Реализация рассмотренных алгоритмов

13.1 Алгоритм построения эквивалентного замыкания бинарного отношения.

– **Построение исходного замыкания.**

Программный код:

```
#include <iostream>
#include <windows.h>
#include <vector>

using namespace std;

int main(){

    SetConsoleOutputCP(CP_UTF8);

    int n;
    cin >> n;
    vector<vector<int>> m(n, vector<int> (n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    vector<pair<int, int>> a;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (m[i][j] == 1) {
                a.push_back({i + 1, j + 1});
            }
        }
    }

    for (int i = 0; i < size(a); ++i) {
        if (i > 0) {
            cout << ", ";
        }
        cout << "(" << a[i].first << ", " << a[i].second << ")";
    }

    return 0;
}
```

Результаты тестирования:

```
5
0 1 0 1 1
1 0 0 0 0
0 1 1 1 0
0 1 1 0 0
0 0 0 1 0
(1, 2), (1, 4), (1, 5), (2, 1), (3, 2), (3, 3), (3, 4), (4, 2), (4, 3), (5, 4)
```

- **Построение эквивалентного замыкания.**

Программный код:

```
#include <iostream>
#include <windows.h>
#include <vector>
#include <set>

using namespace std;

bool transitivity(vector<vector<int>>& m) {

    vector<vector<int>> w(size(m), vector<int> (size(m)));

    for (int i = 0; i < size(m); ++i) {
        for (int j = 0; j < size(m); ++j) {
            w[i][j] = 0;
        }
    }

    for (int i = 0; i < size(m); ++i) {
        for (int l = 0; l < size(m); ++l) {
            int s = 0;
            for (int j = 0; j < size(m); ++j) {
                s += m[i][j] * m[j][l];
            }
            w[i][l] += s;
            if (w[i][l] > 1) {
                w[i][l] = 1;
            }
        }
    }

    bool q = true;

    for (int i = 0; i < size(m); ++i) {
        for (int j = 0; j < size(m); ++j) {
            if (w[i][j] > m[i][j]) {
                q = false;
            }
        }
    }

    return q;

}

int main(){

    SetConsoleOutputCP(CP_UTF8);
```

```
    int n;
    cin >> n;
    vector<vector<int>> m(n, vector<int> (n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
        }
    }

    vector<pair<int, int>> a;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (m[i][j] == 1) {
                a.push_back({i + 1, j + 1});
            }
        }
    }

    for (int i = 0; i < n; ++i) {
        if (m[i][i] == 0) {
            a.push_back({i + 1, i + 1});
        }
    }

    int q = size(a);

    for (int i = 0; i < q; ++i) {
        a.push_back({a[i].second, a[i].first});
    }

    vector<vector<int>> w(n, vector<int> (n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            w[i][j] = 0;
        }
    }

    for (int i = 0; i < n; ++i) {
        for (int l = 0; l < n; ++l) {
            int s = 0;
            for (int j = 0; j < n; ++j) {
                s += m[i][j] * m[j][l];
            }
            w[i][l] += s;
            if (w[i][l] > 1) {
                w[i][l] = 1;
            }
        }
    }

vector<vector<int>> copy_m(size(m), vector<int> (size(m)));

    for (int i = 0; i < size(m); ++i) {
        for (int j = 0; j < size(m); ++j) {
            copy_m[i][j] = m[i][j];
        }

    }
```

```
    while (!transitivity(copy_m)) {
        for (int i = 0; i < size(m); ++i) {
            for (int j = 0; j < size(m); ++j) {
                if (copy_m[i][j] == 0 && w[i][j] == 1) {
                    a.push_back({i + 1, j + 1});
                    copy_m[i][j] = 1;
                }
            }
        }
        if (!transitivity(copy_m)) {
            for (int i = 0; i < size(m); ++i) {
                for (int l = 0; l < size(m); ++l) {
                    int s = 0;
                    for (int j = 0; j < size(m); ++j) {
                        s += copy_m[i][j] * m[j][l];
                    }
                    w[i][l] += s;
                    if (w[i][l] > 1) {
                        w[i][l] = 1;
                    }
                }
            }
        }
    }

    set<pair<int, int>> rrr(begin(a), end(a));

    bool isFirst = true;
    for (auto [i, j] : rrr) {
        if (!isFirst) {
            cout << ", ";
        }
        cout << "(" << i << ", " << j << ")";
        isFirst = false;
    }

    return 0;
}
```

Результаты тестирования:

```
5
0 0 0 1 0
0 0 1 0 0
0 0 1 1 0
1 0 0 0 0
0 0 0 1 1
(1, 1), (1, 4), (2, 2), (2, 3), (2, 4), (3, 1), (3, 2), (3, 3), (3, 4), (4,
 1), (4, 3), (4, 4), (4, 5), (5, 1), (5, 4),
(5, 5)
```

13.2 Алгоритм построения системы представителей фактор-множества.

Программный код:

```
#include <iostream>
```

```cpp
#include <windows.h>
#include <vector>
#include <algorithm>

using namespace std;


void dfs1(const vector<vector<int>>& graph, vector<bool>& used, vector<int>&
order, int v) {
    used[v] = true;
    for (int i = 0; i < size(graph[v]); ++i){
        if (graph[v][i] && !used[i]) {
            dfs1(graph, used, order, i);
        }
    }
    order.push_back(v);
}

void dfs2(const vector<vector<int>>& graph_tr, vector<bool>& used, vector<int>&
component, int v) {
    used[v] = true;
    component.push_back(v);
    for (int i = 0; i < size(graph_tr[v]); ++i) {
        if (graph_tr[v][i] && !used[i]) {
            dfs2(graph_tr, used, component, i);
        }
    }
}

int main() {

    SetConsoleOutputCP(CP_UTF8);

    cout << "Введите n:" << endl;
    int n;
    cin >> n;

    cout << "Введите матрицу n*n:" << endl;
    vector<vector<int>> m(n, vector<int> (n));
    vector<vector<int>> m_tr(n, vector<int> (n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> m[i][j];
            if (m[i][j]) {
                m_tr[j][i] = 1;
            }
        }
    }

    vector<bool> used(n);
    vector<int> order;

    for (int i = 0; i < n; ++i) {
        if (!used[i]) {
            dfs1(m, used, order, i);
        }
    }

    used.assign(n, false);
    vector<int> result;

    for (int i = n - 1; i >= 0; --i) {
        if (!used[i]) {
```

```
            vector<int> component;
            dfs2(m_tr, used, component, i);
            result.push_back(component[0] + 1);
        }
    }

    sort(begin(result), end(result));

    cout << "Представители фактор-множества:" << endl;
    for (int element : result) {
        cout << element << ' ';
    }
    cout << endl;

    return 0;
}
```

Результаты тестирования:

```
Введите n:
5
Введите матрицу n*n:
 1 0 0 0 0
 0 1 0 1 0
 0 0 1 0 0
 0 1 0 1 0
 0 0 0 0 1
Представители фактор-множества:
1 3 4 5
```

13.3 Алгоритм вычисления минимальных (максимальных) и наименьших (наибольших) элементов.

Программный код:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <windows.h>
#include <set>


using namespace std;

int main() {

    SetConsoleOutputCP(CP_UTF8);

    cout << "Введите число x:" << endl;
    int x;
    cin >> x;
    vector<int> divisor_set;

    for (int i = 1; i <= x; ++i) {
        if (x % i == 0) {
```

```
        divisor_set.push_back(i);
    }
}

cout << "Множество делителей x:" << endl;
for (int element : divisor_set) {
    cout << element << ' ';
}

cout << endl << "Наименьший и наибольший элементы: " << endl;
for (int element : divisor_set) {
    if (element == 1 || element == x) {
        if (element == 1) {
            cout << element << " - наименьший элемент" << endl;
        } else {
            cout << element << " - наибольший элемент" << endl;
        }
    }
}

auto iter_first = begin(divisor_set);
divisor_set.erase(iter_first);
divisor_set.pop_back();

int q = x;
int w = 2;
vector<int> simple_divisors;

while (q != 1) {
    if (q % w == 0) {
        simple_divisors.push_back(w);
        q = q / w;
    } else ++w;
}

if (!empty(simple_divisors)) {

    set<int> minimal_elements(begin(simple_divisors), end(simple_divisors));

    cout << "Минимальные элементы:" << endl;
    for (int element : minimal_elements) {
        cout << element << ' ';
    }
    cout << endl;

} else {
    cout << "Минимальный элемент - 1" << endl;
}

if (!empty(simple_divisors)) {

    set<int> maximum_elements;
    for (int i = 0; i < size(simple_divisors); ++i) {
        maximum_elements.emplace(x / simple_divisors[i]);
    }

    cout << "Максимальные элементы:" << endl;
    for (int element : maximum_elements) {
        cout << element << ' ';
    }
    cout << endl;

} else {
```

```
        cout << "Максимальный элемент -" << x;
    }

    return 0;
}
```

Результат тестирования:

```
Введите число x:
40
Множество делителей x:
1 2 4 5 8 10 20 40
Наименьший и наибольший элементы:
1 - наименьший элемент
40 - наибольший элемент
Минимальные элементы:
2 5
Максимальные элементы:
8 20
```

13.4 Алгоритм построения диаграммы Хассе.

– Алгоритм диаграммы Хассе для числа

Программный код:

```
#include <iostream>
#include <windows.h>
#include <vector>
#include <map>
#include <set>
#include <algorithm>

using namespace std;

void dfs(int x, int cur_level, map<int, bool>& used, map<int, int>& levels,
map<int, set<int>>& connect) {
    used[x] = true;
    levels[x] = cur_level;

    for (int d = 1; d <= x; ++d) {

        if (x % d == 0) {
            if (!used[x / d]) {
                connect[x].insert(x / d);
                dfs(x / d, cur_level + 1, used, levels, connect);
            } else if (levels[x / d] == cur_level + 1) {
                connect[x].insert(x / d);
            }
        }
    }
}

int main() {
```

```
    SetConsoleOutputCP(CP_UTF8);

    cout << "Введите число x:" << endl;
    int x;
    cin >> x;

    map<int, bool> used;
    map<int, int> levels;
    map<int, set<int>> connect;

    dfs(x, 0, used, levels, connect);

    vector<pair<int, int>> res(begin(levels), end(levels));

    sort(begin(res), end(res), [](const pair<int, int>& lhs, const pair<int,
int>& rhs) {
        return lhs.second < rhs.second
               || lhs.second == rhs.second && lhs.first < rhs.first;
    });

    cout << "Диаграмма Хассе:" << endl;
    for (auto [k, v] : res) {
        cout << "Число: " << k << ' ' << "Уровень: " << v << ' ';
        cout << "Связано с: ";
        for (auto to : connect[k]) {
            cout << to << ' ';
        }
        cout << endl;
    }

}
```

Результат тестирования:

```
Введите число x:
40
Диаграмма Хассе:
Число: 40 Уровень: 0 Связано с: 8 20
Число: 8 Уровень: 1 Связано с: 4
Число: 20 Уровень: 1 Связано с: 4 10
Число: 4 Уровень: 2 Связано с: 2
Число: 10 Уровень: 2 Связано с: 2 5
Число: 2 Уровень: 3 Связано с: 1
Число: 5 Уровень: 3 Связано с: 1
Число: 1 Уровень: 4 Связано с:
```

– **Алгоритм диаграммы Хассе для матрицы.**

Программный код:

```
#include <iostream>
#include <windows.h>
#include <vector>
#include <set>
```

```
#include <algorithm>

using namespace std;

vector<int> get_min_matrix(const vector<vector<int>>& matrix, const set<int>&
elements) {
    vector<int> minimal;

    for (int i = 0; i < size(matrix); ++i) {
        bool flag = true;

        for (int j = 0; j < size(matrix); ++j)
            if (matrix[j][i] && i != j) {
                flag = false;
                break;
            }

        if (flag && elements.count(i)) {
            minimal.push_back(i);
        }
    }

    return minimal;
}

void remove_min_elements(set<int>& elements, const vector<int>& minimal) {
   for (auto elem : minimal) {
       elements.erase(elem);
   }
}

void clean_rows(vector<vector<int>>& matrix, const vector<int>& minimal)
{
    for (int min : minimal)
        matrix[min].assign(size(matrix), 0);
}

void get_hasse_connections_matrix(const vector<vector<int>>& matrix,
                                  const vector<int>& p_level,
                                  const vector<int>& n_level,
                                  vector<vector<pair<int, int>>>&
levels_connections) {

    vector<pair<int, int>> level_connections;

    for (int p_level_element : p_level) {
        for (int n_level_element: n_level) {
            if (matrix[p_level_element][n_level_element] == 1) {
                level_connections.push_back({p_level_element + 1,
n_level_element + 1});
            }
        }
    }
    levels_connections.push_back(level_connections);
}

void levels_and_connections (vector<vector<int>> hasse_levels,
vector<vector<pair<int, int>>> hasse_level_connections) {

    cout << "Диаграмма Хассе по уровням:" << endl;

    for (int i = size(hasse_levels) - 1; i >= 0; --i) {
        for (int j = 0; j < hasse_levels[i].size (); ++j) {
```

```
            cout << hasse_levels[i][j] + 1 << " ";
        }
        cout << endl;
    }

    cout << "Связи в диаграмме Хассе:" << endl;

    for (int i = hasse_level_connections.size () - 1; i >= 0; --i) {
        for (int j = 0; j < size(hasse_level_connections[i]); ++j) {
            auto level_element = hasse_level_connections[i][j];

            cout << "( " << level_element.first << " связан с " <<
level_element.second << ")" << (j == size(hasse_level_connections[i]) - 1 ? "."
: ", ");
        }
        cout << endl;
    }
}

int main() {

    SetConsoleOutputCP(CP_UTF8);

    cout << "Введите n:" << endl;
    int n;
    cin >> n;

    cout << "Введите матрицу n*n:" << endl;
    vector<vector<int>> matrix(n, vector<int>(n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> matrix[i][j];
        }
    }

    vector<vector<int>> original_matrix = matrix;

    set<int> elements;
    for (int i = 0; i < size(matrix); ++i) {
        elements.insert(i);
    }

    vector<vector<int>> hasse_levels;

    while (!empty(elements)) {
        vector<int> minimal = get_min_matrix(matrix, elements);
        hasse_levels.push_back(minimal);
        remove_min_elements(elements, minimal);
        clean_rows(matrix, minimal);
    }

    vector<vector<pair<int, int>>> hasse_level_connections;
    for (int i = 0; i < size(hasse_levels) - 1; ++i) {
        get_hasse_connections_matrix(original_matrix, hasse_levels[i],
hasse_levels[i + 1], hasse_level_connections);
    }

    levels_and_connections(hasse_levels, hasse_level_connections);

    return 0;
}
```

Результат тестирования:

```
Введите n:
9
Введите матрицу n*n:
1 0 1 0 1 1 1 1 1
0 1 1 0 0 1 1 0 1
0 0 1 0 0 1 1 0 1
0 0 0 1 1 0 1 1 1
0 0 0 0 1 0 1 1 1
0 0 0 0 0 1 1 0 1
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 1
Диаграмма Хассе по уровням:
7 9
6 8
3 5
1 2 4
Связи в диаграмме Хассе:
( 6 связан с 7), ( 6 связан с 9), ( 8 связан с 7), ( 8 связан с 9).
( 3 связан с 6), ( 5 связан с 8).
( 1 связан с 3), ( 1 связан с 5), ( 2 связан с 3), ( 4 связан с 5).
```

13.5 Алгоритм вычисления решётки концептов.

Программный код:

```
#include <iostream>
#include <windows.h>
#include <vector>
#include <set>
#include <algorithm>
#include <numeric>

using namespace std;


set<int> inter_set(const set<int>& a, const set<int>& b) {
    vector<int> res;
    set_intersection(begin(a), end(a), begin(b), end(b), back_inserter(res));

    return {begin(res), end(res)};
}

int main() {

    SetConsoleOutputCP(CP_UTF8);

    cout << "Введите количество объектов и количество атрибутов:" << endl;
    int n;
    cin >> n;

    cout << "Введите матрицу:" << endl;
    vector<vector<int>> matrix(n, vector<int>(n));
    vector<set<int>> a(n);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> matrix[i][j];
            if (matrix[i][j]) {
```

```
                a[j].insert(i);
            }
        }
    }

    set<set<int>> inter(begin(a), end(a));

    vector<int> full(n);
    iota(begin(full), end(full), 0);

    inter.insert(set<int>(begin(full), end(full)));
    inter.insert(set<int>());

    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            inter.insert(inter_set(a[i], a[j]));
        }
    }

    vector<pair<vector<int>, vector<char>>> ans;

    for (const auto& s : inter) {
        ans.push_back({});
        for (int i = 0; i < n; ++i) {
            if (size(inter_set(s, a[i])) == size(s)) {
                ans.back().second.push_back('a' + i);
            }
        }
        ans.back().first = vector<int>(begin(s), end(s));
    }

    sort(begin(ans), end(ans), [](const auto& lhs, const auto& rhs) {return
size(lhs.second) > size(rhs.second);});


    int level = 0;
    int i = 0;

    cout << "Итоговый результат:" << endl;
    for (const auto& [s, names] : ans) {
        cout << "[ ";
        for (auto e : s) {
            cout << e + 1 << ' ';
        }
        cout << "] ";

        cout << "[ ";
        for (auto name : names) {
            cout << name << ' ';
        }
        cout << "] ";

        if (i == 0 || size(ans[i].second) != size(ans[i - 1].second)) {
            ++level;
        }

        cout << "Уровень: " << level << endl;

        ++i;
    }
```

```
    return 0;
}
```

Результат тестирования:

```
Введите количество объектов и количество атрибутов:
4
Введите матрицу:
 1 0 1 0
 1 1 0 0
 0 1 0 1
 0 1 0 1
Итоговый результат:
[ ] [ a b c d ] Уровень: 1
[ 1 ] [ a c ] Уровень: 2
[ 2 ] [ a b ] Уровень: 2
[ 3 4 ] [ b d ] Уровень: 2
[ 1 2 ] [ a ] Уровень: 3
[ 2 3 4 ] [ b ] Уровень: 3
[ 1 2 3 4 ] [ ] Уровень: 4
```

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены теоретические сведения об отношении эквивалентности, разобраны определения фактор-множества, отношения порядка и диаграммы Хассе.

Результатом работы является: алгоритм построения эквивалентного замыкания бинарного отношения, алгоритм построения системы представителей фактор-множества, алгоритм вычисления минимальных (максимальных) и наименьших (наибольших) элементов, алгоритм построения диаграммы Хассе и алгоритм вычисления решётки концептов.

Была осуществлена программная реализация описанных алгоритмов и проведено тестирование программ.