МИНОБРНАУКИ РОССИИ
ФГБОУ ВО «СГУ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

ОТНОШЕНИЕ ЭКВИВАЛЕНТНОСТИ И ОТНОШЕНИЕ ПОРЯДКА

ЛАБОРАТОРНАЯ РАБОТА

студента 3 курса 331 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Токарева Никиты Сергеевича

Проверил
аспирант ____________ В. Н. Кутин

Саратов 2022

1 Постановка задачи

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 Теоретические сведения по рассмотренным темам с их обоснованием

2.1 Определение отношения эквивалентности и фактор-множества

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

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

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

Срезы $\varepsilon(a)$ называются классами эквивалентности по отношению ε и сокращенно обозначаются символом $[a]$. Множество всех таких классов эквивалентности $\{[a] : a \in A\}$ называется фактор-множеством множества A по эквивалентности ε и обозначается символом A/ε.

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

— $\varepsilon(T) = A$;

— 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.

Лемма 1. О замыканиях бинарных отношений.

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. $f_t(\rho) = \bigcup_{n=1}^{\infty} \rho^n$ - наименьшее транзитивное бинарное отношение, содержащее отношение $\rho \subset A^2$,
4. $f_{eq}(\rho) = f_t f_s f_r(\rho)$ - наименьшее отношение эквивалентности, на содержащее отношение $\rho \subset A^2$.

2.2 Определение отношения порядка

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

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

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

1. минимальным, если $(\forall x \in A)\ x \leq a \implies x = a$,
2. максимальным, если $(\forall x \in A)\ a \leq x \implies x = a$,
3. наименьшим, если $(\forall x \in A)\ a \leq x$,
4. наибольшим, если $(\forall x \in A)\ x \leq a$.

Лемма 2.

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 Определение диаграммы Хассе

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

Стоит отметить, что 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.

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

1. В упорядоченном множестве $A = (A, \leq)$ найти множество A_1 всех минимальных элементов и расположить их в один горизонтальный ряд (это первый уровень диаграммы).
2. В упорядоченном множестве $A \setminus A_1$, найти множество A_2 всех минимальных элементов и расположить их в один горизонтальный ряд над первым уровнем (это второй уровень диаграммы). Соединить отрезками элементы

этого ряда с покрываемыми ими элементами предыдущего ряда.

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

2.4 Определение контекста и концепта

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

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

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

Алгоритм вычисления системы замыканий на множестве G:

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

3 Результаты работы

3.1 Описание алгоритма построения эквивалентного замыкания бинарного отношения и системы представителей фактор-множества

Алгоритм 1 - Построение эквивалентного замыкания

Вход: Матрица бинарного отношения $A = (a_{ij})$ размерности $n \times n$.

Выход: Исходная матрица бинарного отношения, замкнутая относительно рефлексивности, симметричности и транзитивности.

Шаг 1. Построить рефлексивное замыкание на матрице $A = (a_{ij})$ бинарного отношения. Необходимо пройти по всем элементам a_{ii} матрицы A, где $0 \leq i < n$, и присвоить им единицу (т.е. $a_{ii} = 1$).

Шаг 2. Выполнив шаг 1, построить симмтеричное замыкание на матрице A бинарного отношения. Необходимо пройти по всем элементам a_{ij} матрицы A, где $0 \leq i, j < n$ Если элемент $a_{ij} = 1$, а элемент $a_{ji} \neq 1$, то нужно присвоить a_{ji} единицу.

Шаг 3. Выполнив шаг 2, построить транзитивное замыкание на матрице A бинарного отношения. Если $a_{ik} = 1 \wedge a_{kj} = 1$, то нужно приравнять a_{ij} к единице, где a_{ik}, a_{kj}, a_{ij} – элементы матрицы A, $(0 \leq i, j, k < n)$. Данная процедура должна повторится n раз согласно пункту 3 леммы 1 (о замыканиях бинарных отношений), где говорится об операторе транзитивного замыкания.

Шаг 4. Вернуть полученную матрицу $A = (a_{ij})$ бинарного отношения, замкнутой относительно рефлексивности, симметричности и транзитивности.

Оценка сложности данного алгоритма равна $O(n + n^2 + n^4) = O(n^4)$.

Алгоритм 2 - Построение системы представителей фактор-множества

Вход: Матрица бинарного отношения $A = (a_{ij})$ размерности $n \times n$.

Выход: Список $repsys$, в котором содержатся числа, составляющие систему представителей T фактор-множества X/ε бинарного отношения на множестве X.

Шаг 1. Создать пустые списоки $tmp = []$, $fset = []$ и $repsys = []$.

Шаг 2. Необходимо построить фактор-множество X/ε, получив срезы по каждому элементу $a \in X$. Для этого нужно проверить элементы a_{ij} матрицы A. Если $a_{ij} = 1$, где $0 \leq i, j < n$, то добавить значение j в список tmp.

Шаг 3. Если полученного списка tmp нет в списке $fset$, то добавить его в список

$fset$, в противном случае – не добавлять.
Шаг 4. Найти в каждом подсписке $fset[i]$ $(0 \leq i < k)$ списка $fset$ минимум el_{min}, и затем добавить в список $repsys$ пару $(el_{min}, fset[i])$.
Шаг 5. Вернуть список $repsys$ в качестве выхода функции.

Оценка сложности данного алгоритма состоит из оценки сложности формирования списка $fset$ равной $O(n^2)$ и оценки сложности нахождения минимума для формирования списка $repsys$ равной $O(n^2)$. В итоге получаем оценку: $O(n^2 + n^2) = O(n^2)$.

3.2 Описание алгоритмов вычисления минимальных (максимальных) и наименьших (наибольших) элементов и построения диаграммы Хассе

Алгоритм 3 - Вычисление минимального элемента множества

Вход: Матрица бинарного отношения $A = (a_{ij})$ размерности $n \times n$ и список set_x, состоящий из элементов множества X, и число k.
Выход: Список L_{min}, значения которого являются минимальными элементами упорядоченного множества $(X, \leq)$.
Шаг 1. Создать пустой список $L_{min} = []$.
Шаг 2. Создать цикл с индексом i, где $k \leq i < n$, в котором инициализировать булеву переменную $fl = true$. Затем создать вложенный цикл с индексом j, где $k \leq j < n$, в котором поставить условие. Если $a[i][j] \neq 0 \wedge i \neq j$, то булевой переменной присвоить $fl = false$.
Шаг 3. Если после завершения вложенного цикла с индексом j значение флага не изменилось (т.е. $fl = true$), то добавить в список L_{min} значение $set_x[j]$.
Шаг 4. Вернуть список L_{min} в качестве выхода функции.

Оценка сложности данного алгоритма равна $O(n^2)$.

Алгоритм 4 - Вычисление максимального элемента множества

Вход: Матрица бинарного отношения $A = (a_{ij})$ размерности $n \times n$ и список set_x, состоящий из элементов множества X, и число k.
Выход: Список L_{max}, значения которого являются максимальными элементами упорядоченного множества $(X, \leq)$.
Шаг 1. Создать пустой список $L_{max} = []$.

Шаг 2. Создать цикл с индексом i, где $k \leq i < n$, в котором инициализировать булеву переменную $fl = true$. Затем создать вложенный цикл с индексом j, где $k \leq j < n$, в котором поставить условие. Если $a[j][i] \neq 0 \wedge i \neq j$, то булевой переменной присвоить $fl = false$.
Шаг 3. Если после завершения вложенного цикла с индексом j значение флага не изменилось (т.е. $fl = true$), то добавить в список L_{max} значение $set_x[j]$.
Шаг 4. Вернуть список L_{max} в качестве выхода функции.

Оценка сложности данного алгоритма равна $O(n^2)$.

Алгоритм 5 - Вычисление наименьшего элемента множества

Вход: Список L_{min}, значения которого являются минимальными элементами упорядоченного множества $(X, \leq)$.
Выход: Наименьший элемент x_{min} упорядоченного множества $(X, \leq)$ или значение -1.
Шаг 1. Если длина L_{min} не равна 1, то вернуть -1, иначе – вернуть единственный элемент $x_{min} = l \in L_{min}$, являющийся наименьшим элементом множества X.

Трудоемкость алгоритма $O(1)$.

Алгоритм 6 - Вычисление наибольшего элемента множества

Вход: Список L_{max}, значения которого являются максимальными элементами упорядоченного множества $(X, \leq)$.
Выход: Наибольший элемент x_{max} упорядоченного множества $(X, \leq)$ или значение -1.
Шаг 1. Если длина L_{max} не равна 1, то вернуть -1, иначе – вернуть единственный элемент $x_{max} = l \in L_{max}$, являющийся наибольшим элементом множества X.

Трудоемкость алгоритма $O(1)$.

Алгоритм 7 - Построение диаграммы Хассе

Вход: Матрица бинарного отношения порядка $A = (a_{ij})$ размерности $n \times n$ и список set_x, состоящий из элементов множества X.
Выход: Список H длиной n, характеризующий диаграмму Хассе: каждый эле-

мент в списке представляет собой четыре значения: элемент $x \in X$, значение его уровня l на диаграмме, список $prevV$ элементов множества X, находящихся на уровне $lvl - 1$, список $nextV$ элементов множества X, находящихся на уровне $lvl + 1$ относительно элемента x.

Шаг 1. Инициализировать списки $diagramm = []$, $el\_and\_lvl = []$ и переменные $k = 0$, $lvl = 0$.

Шаг 2. Создать цикл, с условием, что пока $k \neq n$, где n – длина списка set_x. Запустить алгоритм 3, где в качестве входных данных подать матрицу A, список set_x и число k.

Шаг 3. При завершении алгоритма 3 был получен список L_{min}, значения которого являются минимальными элементами упорядоченного множества $(X, \leq)$. Далее нужно увеличить значение переменной lvl на 1.

Шаг 4. Необходимо создать цикл, где совершается проход по всем элементам el списка L_{min}, для того чтобы сформировать пару (el, lvl) и добавить её в список el_and_lvl. При добавлении нового элемента в список el_and_lvl увеличивается значение переменной k на 1.

Шаг 5. После завершения цикла, созданного на шаге 2, необходимо создать цикл с индексом i $(0 \leq i < n)$, в котором инициализировать списки $nextV = []$ и $prevV = []$ и переменные $nextlvl = el\_and\_lvl[i][1] + 1$ и $prevlvl = el\_and\_lvl[i][1] - 1$. Далее нужно создать вложенный цикл с индексом j $(0 \leq j < n)$, где выполняются следующие условия. Если $el\_and\_lvl[j][1] = nextlvl$, то добавить в список $nextV$ элемент $el\_and\_lvl[j][0]$. Если $el\_and\_lvl[j][1] = prevlvl$, то добавить в список $prevV$ элемент $el\_and\_lvl[j][0]$.

Шаг 6. С учетом предыдущих шагов, нужно добавить в список $diagramm$ следующий кортеж: $(el\_and\_lvl[j][0], el\_and\_lvl[j][1], prevV, nextV)$.

Шаг 7. Вернуть полученный список $diagramm$ в качестве выхода функции.

Оценка сложности данного алгоритма с учётом оценки сложности построения списка el_and_lvl равной $O(n^2)$ и с учетом оценки сложности построения списка $diagramm$ равной $O(n^2)$ будет равна $O(n^2 + n^2) = O(n^2)$.

3.3 Описание алгоритма построения решетки концептов

Алгоритм 8 - Построение системы замыканий

Вход: Список $objs$, который содержит n элементов множества объектов G, спи-

сок $attrs$, который содержит k элементов множества объектов M и матрица $A = (a_{ij})$ бинарного отношения $\rho \subset G \times M$ размерности $n \times k$.
Выход: Список set_z, значения которого являются элементы системы замыканий Z_{f_G} на множестве G.
Шаг 1. Создать список set_z и добавить в него список $objs$ в качестве первого элемента.
Шаг 2. Создать цикл с индексом i, где $0 \leq i < k$, в котором инициализировать пустой список $cur\_slice = []$. Затем создать вложенный цикл с индексом j, где $0 \leq j < n$, в котором поставить условие. Если $a[j][i] = 1$, то добавить в список cur_slice значение $objs[j]$.
Шаг 3. Далее необходимо сделать пересечение полученного множества cur_slice с каждым элементом из множества set_z. Обозначим текущее пересечение, как $x \in set_z$. Если элемент x не содержится в данном множестве (списке) set_z, то его следует добавить в список set_z, иначе пропустить.
Шаг 4. Вернуть список set_z в качестве выхода функции.

Оценка сложности данного алгоритма равна $O(n \cdot k + n^2)$.

Алгоритм 9 - Построение решетки концептов

Вход: Список set_z, значения которого являются элементы системы замыканий Z_{f_G} на множестве G. Список $objs$, содержащий n элементов множества объектов G, список $attrs$, который содержит k элементов множества объектов M и матрица $A = (a_{ij})$ бинарного отношения $\rho \subset G \times M$ размерности $n \times k$.
Выход: список $set\_isomorph$, который содержит элементы решетки концептов $(C(K), \leq)$.
Шаг 1. Инициализировать словарь $tmp\_iso =$, где ключами будут являться элементы $objs[i]$ $(0 \leq i < n)$, а значениями – список элементов, который формируется на шаге 2.
Шаг 2. Создать цикл с индексом i, где $0 \leq i < n$, в котором инициализировать пустой список $cur\_slice = []$. Далее создать вложенный цикл с индексом j, где $0 \leq j < k$, в котором поставить условие. Если $a[j][i] = 1$, то добавить в список cur_slice значение $attrs[j]$. Затем необходимо добавить в словарь элемент, где ключом будет являться $objs[i]$, а значением – список cur_slice.
Шаг 3. Инициализировать список $set\_isomorph = []$. Далее, проходя по всем спискам el, принадлижащих списку set_z, производится пересечение элементов

$tmp\_iso[el[i]]$ ($0 \leq i < l$, где l - длина списка el), которую обозначим $intersec$.
Шаг 4. Формируем пару (arg_1, arg_2), где arg_1 – текущий список el, arg_2 – пересечение $intersec$. Такую пару добавляем в список $set\_isomorph$. В конечном итоге получаются элементы решетки концептов $(C(K), \leq)$.
Шаг 5. Вернуть полученный список $set\_isomorph$ в качестве выхода функции.

Оценка сложности данного алгоритма состоит из оценки сложности прохождения по элементам a_{ij} матрицы A ($0 \leq i < n$, $0 \leq j < k$), равной $O(n \cdot k)$ и оценки сложности прохождения по всем элементам списка set_z,т.е. $O(m \cdot n)$, где m – количество элементов списка set_z. Тогда оценка сложности равна: $O(n \cdot (k + m))$.

3.4 Коды программ, реализующей рассмотренные алгоритмы

```
import matplotlib.pyplot as plt

# Построение матрицы по бинарному отношению
def go_to_matrix(br, n):
    n = -1
    for pair in br:
        tmp = max(pair)
        n = max(tmp, n)
    n += 1
    a = []
    for i in range(n):
        a.append([0 for j in range(n)])
    for pair in br:
        a[pair[0]][pair[1]] = 1
    return np.array(a)


# Построение бинарного отношения по матрице
def go_to_set(a, n):
    s = []
    for i in range(n):
        for j in range(n):
            if a[i][j] == 1:
                s.append((i, j))
    return s
```

```python
# Построение эквивалентности
def get_equivalence(a, n):
    for i in range(n):
        a[i][i] = 1
        for j in range(n):
            if a[i][j] == 1 and a[i][j] != a[j][i]:
                a[j][i] = 1
    for m in range(n):
        for i in range(n):
            for j in range(n):
                for k in range(n):
                    if a[i][k] and a[k][j]:
                        a[i][j] = 1
    return


# Построение фактор-множества
def get_factor_set(a, n):
    s = []
    for i in range(n):
        tmp = []
        for j in range(n):
            if a[i][j]:
                tmp.append(j)
        if tmp not in s:
            s.append(tmp)
    return s


# Построение системы представителей классов
def get_representative_system(fset, n):
    repsys = []
    for i in range(n):
        repsys.append((min(fset[i]), fset[i]))
    return repsys


# Построение множества общих делителей
def get_set_of_common_dividers(num, exptn):
```

```
    dvdrs = []
    for i in range(1, int(num / 2) + 1):
        if num % i == 0 and i not in exptn:
            dvdrs.append(i)
    if num not in exptn:
        dvdrs.append(num)
    return dvdrs


# Построение списка элементов и их общих делителей
def get_level_list(dvdrs):
    div_dict = {}
    for el in dvdrs:
        dividers_list = []
        for div in dvdrs:
            if el % div == 0:
                dividers_list.append(div)
        div_dict[el] = (dividers_list)
    return div_dict


# Нахождение минимальных элементов
def get_minimal_elem(slices_list, is_div_mode, set_x, strt=0):
    if is_div_mode:
        minel_len = len(slices_list[set_x[0]])
        min_el_list = []
        for key in slices_list:
            tmp = len(slices_list[key])
            if minel_len > tmp:
                minel_len = tmp
        for key in slices_list:
            if minel_len == len(slices_list[key]):
                min_el_list.append(key)
        return min_el_list
    else:
        min_el_list = []
        for i in range(strt, len(slices_list)):
            fl = True
            for j in range(strt, len(slices_list[i])):
                if slices_list[i][j] != 0 and i != j:
                    fl = False
```

```
            if fl:
                min_el_list.append(set_x[i])
        return min_el_list


# Нахождение максимальных элементов
def get_maximal_elem(slices_list, is_div_mode, set_x, strt=0):
    if is_div_mode:
        maxel_len = len(slices_list[set_x[0]])
        max_el_list = []
        for key in slices_list:
            tmp = len(slices_list[key])
            if maxel_len < tmp:
                maxel_len = tmp
        for key in slices_list:
            if maxel_len == len(slices_list[key]):
                max_el_list.append(key)
        return max_el_list
    else:
        max_el_list = []
        for i in range(strt, len(slices_list)):
            fl = True
            for j in range(strt, len(slices_list[i])):
                if slices_list[j][i] != 0 and i != j:
                    fl = False
            if fl:
                max_el_list.append(set_x[i])
        return max_el_list


# Нахождение наименьшего элемента
def get_least_elem(min_elems):
    if len(min_elems) == 1:
        return min_elems[0]
    else:
        return -1


# Нахождение наибольшего элемента
def get_greatest_elem(max_elems):
    if len(max_elems) == 1:
```

```python
            return max_elems[0]
        else:
            return -1


# Получение диаграммы Хассе
def get_diagramm_Hasse(dvdrs_list, is_div_mode, set_x):
    diagramm = []
    lvl = 0
    el_and_lvl = []
    if is_div_mode:
        while dvdrs_list != {}:
            min_elems = get_minimal_elem(dvdrs_list, True, set_x)
            lvl += 1
            for el in min_elems:
                set_x.remove(el)
                el_and_lvl.append([el, lvl])
                del dvdrs_list[el]
                for key in dvdrs_list:
                    if el in dvdrs_list[key]:
                        dvdrs_list[key].remove(el)
    else:
        k = 0
        while k != len(set_x):
            min_elems = get_minimal_elem(dvdrs_list, False, set_x, k)
            lvl += 1
            for el in min_elems:
                el_and_lvl.append([el, lvl])
                k += 1
    tmp_len = len(el_and_lvl)
    for i in range(tmp_len):
        next_lvl = el_and_lvl[i][1] + 1
        prev_lvl = el_and_lvl[i][1] - 1
        arr_next_lvl = []
        arr_prev_lvl = []
        for j in range(tmp_len):
            if el_and_lvl[j][1] == next_lvl:
                arr_next_lvl.append(el_and_lvl[j][0])
            if el_and_lvl[j][1] == prev_lvl:
                arr_prev_lvl.append(el_and_lvl[j][0])
        diagramm.append(( el_and_lvl[i][0], el_and_lvl[i][1]
```

```
                        , arr_prev_lvl, arr_next_lvl) )
    return diagramm



# Нахождение системы замыкания
def get_closure_system(objs, attrs, a, ob_len, at_len):
    set_z = [objs]
    for i in range(at_len):
        cur_slice = []
        for j in range(ob_len):
            if a[j][i]:
                cur_slice.append(objs[j])
            else:
                continue
        for subset in set_z:
            tmp = list(set(cur_slice) & set(subset))
            tmp.sort()
            if tmp not in set_z:
                set_z.append(tmp)
    return set_z


# Нахождение решетки концептов
def get_lattice_of_concepts(set_z, objs, attrs, a, ob_len, at_len):
    set_iso = {}
    for i in range(ob_len):
        cur_slice = []
        for j in range(at_len):
            if a[i][j]:
                cur_slice.append(attrs[j])
        set_iso[objs[i]] = cur_slice
    set_isomorph = []
    for el in set_z:
        tmp = set([])
        for i in range(len(el)):
            if i == 0:
                tmp = set(set_iso[el[i]])
            else:
                tmp = tmp & set(set_iso[el[i]])
        if el == []:
```

```
            set_isomorph.append([el, attrs])
        else:
            tmp = list(tmp)
            tmp.sort()
            set_isomorph.append([el, tmp])

    return set_isomorph


# Вывод матрицы
def print_matrix(a, n):
    cnt = 0
    print('Your matrix:')
    for i in a:
        if (cnt < n):
            print(f'{i}' + ' ')
        else:
            print(f'{i}' + '\n')
            cnt = 0
        cnt += 1


# Вывод бинарного отношения
def print_binary_relation(br, n):
    print('Your binary relation:')
    print('{ ', end='')
    for i in range(n):
        if i == n - 1:
            print('(' + str(br[i][0] + 1) + ', ' + str(br[i][1] + 1), end=')')
        else:
            print('(' + str(br[i][0] + 1) + ', ' + str(br[i][1] + 1), end='), ')
    print(' }')


# Вывод фактор-множества
def print_factor_set(s, n):
    print('Factor-set:')
    print('{ ', end='')
    for i in range(n):
        print('{', end='')
        m = len(s[i])
```

```python
        for j in range(m):
            if j == m - 1:
                print(str(s[i][j] + 1), end='')
            else:
                print(str(s[i][j] + 1), end=', ')
        if i == n - 1:
            print('}', end='')
        else:
            print('}', end=', ')
    print(' }')


# Вывод системы представителей
def print_representative_system(repsys, n):
    print('Class representative system:')
    elems = []
    sets = []
    print('{ ', end='')
    for i in range(n):
        if i == n - 1:
            print('{' + str(repsys[i][0] + 1) + '}', end='')
        else:
            print('{' + str(repsys[i][0] + 1) + '}, ', end='')
    print(' } ' + 'where ', end='')
    for i in range(n):
        if i == n - 1:
            print('{' + str(repsys[i][0] + 1) + '}', end='')
            print(' from ', end='')
            print('{', end='')
            m = len(repsys[i][1])
            for j in range(m):
                if j == m - 1:
                    print(str(repsys[i][1][j] + 1), end='')
                else:
                    print(str(repsys[i][1][j] + 1), end=', ')
            print('}')
        else:
            print('{' + str(repsys[i][0] + 1) + '}', end='')
            print(' from ', end='')
            print('{', end='')
            m = len(repsys[i][1])
```

```
        for j in range(m):
            if j == m - 1:
                print(str(repsys[i][1][j] + 1), end='')
            else:
                print(str(repsys[i][1][j] + 1), end=', ')
        print('}', end=', ')


def print_Hasse_diagramm(data, is_div_mode, set_list, set_x):
    plt.title("Диаграмма Хассе")

    # Построение ключ: уровень, значение: все элементы
    # на текущем уровне
    el_and_lvl = {}
    for hasse_el in data:
        el_and_lvl[hasse_el[1]] = []
    for hasse_el in data:
        el_and_lvl[hasse_el[1]].append(hasse_el[0])

    lim = len(el_and_lvl) * 2
    plt.xlim(0, lim)
    plt.ylim(0, lim)
    x = {}
    y = {}
    # Нахождение координат элементов
    for key in el_and_lvl:
        step = len(el_and_lvl[key]) + 1
        cur = lim / step
        step = cur
        for el in el_and_lvl[key]:
            x[el] = cur
            y[el] = float(key)
            cur += step
    box = { 'facecolor':'cyan',
        'edgecolor': 'black',
        'boxstyle': 'circle',
        'linewidth': '2' }

    if is_div_mode:
        lines = []
        for elem in data:
```

```
            tmp = []
            for i in range(len(elem[3])):
                if elem[3][i] % elem[0] == 0:
                    tmp.append(elem[3][i])
            for el in tmp:
                lines.append([elem[0], el])

        for line in lines:
            plt.plot([x[line[0]], x[line[1]]], [y[line[0]], y[line[1]]], 'r')

        for key in x:
            plt.text( x[key], y[key], str(key),
                      bbox = box,
                      horizontalalignment = 'center',
                      color = 'black',
                      fontsize = 9 )
        plt.show()
    else:
        lines = []
        for i in range(len(set_list)):
            for j in range(len(set_list[i])):
                if set_list[i][j] and j < i:
                    for el in data[i][2]:
                        if set_x[j] == el:
                            lines.append([data[i][0], set_x[j]])
        for line in lines:
            plt.plot([x[line[0]], x[line[1]]], [y[line[0]], y[line[1]]], 'r')

        for key in x:
            plt.text( x[key], y[key], key,
                      bbox = box,
                      horizontalalignment = 'center',
                      color = 'black',
                      fontsize = 9 )
        plt.show()

# Построение решетки концептов (меню)
def construction_lattice_of_concepts():
    print('Enter set of objects:')
    s = input()
    objs = [i for i in s.split(' ')]
```

```
    print('Enter set of attributes:')
    s = input()
    attrs = [i for i in s.split(' ')]
    print('Enter a binary relation matrix:')
    a = []
    ob_len = len(objs)
    at_len = len(attrs)
    for i in range(ob_len):
        tmp = input().split()
        for j in range(len(tmp)):
            tmp[j] = int(tmp[j])
        a.append(tmp)
    print('Closure system:', end=' ')
    set_z = get_closure_system(objs, attrs, a, len(objs), len(attrs))
    print(set_z)
    print('Lattice of concepts:', end= ' ')
    set_isomorph = get_lattice_of_concepts(set_z, objs, attrs, a, ob_len, at_len)
    print(set_isomorph)

    return choose_mode()

# Построение диаграммы Хассе (меню)
def construction_of_Hasse_diagramm():
    print('Select the way:')
    print('Press 1 to enter some number')
    print('Press 2 to enter set and matrix')
    bl = input()
    if bl == '1':
        print('Enter some number')
        num = int(input())
        print('Enter exception numbers, else press 0')
        s = input()
        exptn = [int(i) for i in s.split(' ')]
        dvdrs = get_set_of_common_dividers(num, exptn)

        dvdrs_list = get_level_list(dvdrs)

        min_elems = get_minimal_elem(dvdrs_list, True, dvdrs)
        max_elems = get_maximal_elem(dvdrs_list, True, dvdrs)
        least_elem = get_least_elem(min_elems)
        greatest_elem = get_greatest_elem(max_elems)
```

```
    print('Minimal elements:', end='{ ')
    for el in min_elems:
        print(el, end=' ')
    print('}')
    print('Maximal elements:', end='{ ')
    for el in max_elems:
        print(el, end=' ')
    print('}')
    print('Least element:', end=' ')
    if least_elem == -1:
        print('None')
    else:
        print('{',least_elem, '}')
    print('Greatest element:', end= ' ')
    if greatest_elem == -1:
        print('None')
    else:
        print('{', greatest_elem, '}')

    print('Do you want to create Hasse\'s diagramm?)')
    print('(1 - yes, 0 - no)')
    bl = input()
    if bl == '1':
        diagramm = get_diagramm_Hasse(dvdrs_list, True, dvdrs)
        print(diagramm)
        print_Hasse_diagramm(diagramm, True, dvdrs_list, dvdrs)
    elif bl == '0':
        return choose_mode()
    else:
        print('Incorrect input!')

elif bl == '2':
    print('Enter some set of elements')
    s = input()
    br = [i for i in s.split(' ')]

    print('Enter the matrix values')
    a = []
    for i in range(len(br)):
        a.append([int(j) for j in input().split()])
    min_elems = get_minimal_elem(a, False, br)
```

```
        max_elems = get_maximal_elem(a, False, br)
        least_elem = get_least_elem(min_elems)
        greatest_elem = get_greatest_elem(max_elems)
        print('Minimal elements:', end='{ ')
        for el in min_elems:
            print(el, end=' ')
        print('}')
        print('Maximal elements:', end=' ')
        for el in max_elems:
            print(el, end=' ')
        print('')
        print('Least element:', end=' ')
        if least_elem == -1:
            print('None')
        else:
            print(least_elem)
        print('Greatest element:', end= ' ')
        if greatest_elem == -1:
            print('None')
        else:
            print(greatest_elem)

        print('Do you want to create Hasse\'s diagramm?)')
        print('(1 - yes, 0 - no)')
        bl = input()
        if bl == '1':
            diagramm = get_diagramm_Hasse(a, False, br)
            print_Hasse_diagramm(diagramm, False, a, br)
        elif bl == '0':
            return choose_mode()
        else:
            print('Incorrect input!')
    else:
        print('Incorrect input!')
    return choose_mode()


# Построение фактор-множества (меню)
def construction_of_factor_sets():
    print('Select the way to specify a binary relation:')
    print('Press 1 to enter matrix')
```

```python
print('Press 2 to enter binary relation elements')
print('Press 3 to exit from programm')
bl = input()
if bl == '1':
    print('Enter the number of verticies')
    n = int(input())
    print('Enter the matrix values')
    a = []
    for i in range(n):
        a.append([int(j) for j in input().split()])

    get_equivalence(a, n)

elif bl == '2':
    print('Enter an even amount of numbers on one line')
    s = input()
    br = [int(i) for i in s.split(' ')]
    if len(br) % 2 != 0:
        print('Incorrect input')
        choose_mode()
    else:
        cnt = 0
        n = max(br)
        a = []
        for i in range(n):
            a.append([0 for j in range(n)])
        for i in range(len(br) // 2):
            a[br[cnt] - 1][br[cnt + 1] - 1] = 1
            cnt += 2

        get_equivalence(a, n)
elif bl == '3':
    return

print_matrix(a, n)
br = go_to_set(a, n)
print_binary_relation(br, len(br))


fset = get_factor_set(a, n)
print_factor_set(fset, len(fset))
```

```python
    repsys = get_representative_system(fset, len(fset))
    print_representative_system(repsys, len(repsys))

    return choose_mode()

# Главное меню
def choose_mode():
    print('Choose mode:')
    print('Press 1 to get factor-set')
    print('Press 2 to create Hasse\'s diagramm')
    print('Press 3 to create a lattice of concepts')
    print('Press 4 to exit')
    bl = input()
    if bl == '1':
        construction_of_factor_sets()
    elif bl == '2':
        construction_of_Hasse_diagramm()
    elif bl == '3':
        construction_lattice_of_concepts()
    elif bl == '4':
        return
    else:
        print('Incorrect output')
        return choose_mode()


choose_mode()
```

3.5 Результаты тестирования программ

```
Choose mode:
Press 1 to get factor-set
Press 2 to create Hasse's diagramm
Press 3 to create a lattice of concepts
Press 4 to exit
1
Select the way to specify a binary relation:
Press 1 to enter matrix
Press 2 to enter binary relation elements
Press 3 to exit from programm
2
Enter an even amount of numbers on one line
1 3 3 4 2 5
Your matrix:
[1 0 1 1 0]
[0 1 0 0 1]
[1 0 1 1 0]
[1 0 1 1 0]
[0 1 0 0 1]
Your binary relation:
{ (1, 1), (1, 3), (1, 4), (2, 2), (2, 5), (3, 1), (3, 3), (3, 4), (4, 1), (4, 3), (4, 4), (5, 2), (5, 5) }
Factor-set:
{ {1, 3, 4}, {2, 5} }
Class representative system:
{ {1}, {2} } where {1} from {1, 3, 4}, {2} from {2, 5}
Choose mode:
Press 1 to get factor-set
Press 2 to create Hasse's diagramm
Press 3 to create a lattice of concepts
Press 4 to exit
```

Рисунок 1 – Тест алгоритма построения эквивалентного замыкания

```
Choose mode:
Press 1 to get factor-set
Press 2 to create Hasse's diagramm
Press 3 to create a lattice of concepts
Press 4 to exit
2
Select the way:
Press 1 to enter some number
Press 2 to enter set and matrix
1
Enter some number
30
Enter exception numbers, else press 0
1
Minimal elements:{ 2 3 5 }
Maximal elements:{ 30 }
Least element: None
Greatest element: { 30 }
Do you want to create Hasse's diagramm?)
(1 - yes, 0 - no)
1
[(2, 1, [], [6, 10, 15]), (3, 1, [], [6, 10, 15]), (5, 1, [], [6, 10, 15]), (6, 2, [2, 3, 5], [30]),
 (10, 2, [2, 3, 5], [30]), (15, 2, [2, 3, 5], [30]), (30, 3, [6, 10, 15], [])]
Choose mode:
Press 1 to get factor-set
Press 2 to create Hasse's diagramm
Press 3 to create a lattice of concepts
Press 4 to exit
```

Рисунок 2 – Тест алгоритма построения диаграммы Хассе на множестве $(X, |)$

```
Press 1 to enter some number
Press 2 to enter set and matrix
2
Enter some set of elements
1 2 3 4 6 9 12 18 36
Enter the matrix values
1 0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0
1 0 1 0 0 0 0 0 0
1 1 0 1 0 0 0 0 0
1 1 1 0 1 0 0 0 0
1 0 1 0 0 1 0 0 0
1 1 1 1 1 0 1 0 0
1 1 1 0 1 1 0 1 0
1 1 1 1 1 1 1 1 1
Minimal elements:{ 1 }
Maximal elements: 36
Least element: 1
Greatest element: 36
Do you want to create Hasse's diagramm?)
(1 - yes, 0 - no)
1
[('1', 1, [], ['2', '3']), ('2', 2, ['1'], ['4', '6', '9']), ('3', 2, ['1'], ['4', '6', '9']), ('4',
3, ['2', '3'], ['12', '18']), ('6', 3, ['2', '3'], ['12', '18']), ('9', 3, ['2', '3'], ['12', '18']),
 ('12', 4, ['4', '6', '9'], ['36']), ('18', 4, ['4', '6', '9'], ['36']), ('36', 5, ['12', '18'], [])]
Choose mode:
Press 1 to get factor-set
Press 2 to create Hasse's diagramm
Press 3 to create a lattice of concepts
Press 4 to exit
```

Рисунок 3 – Тест алгоритма построения диаграммы Хассе на множестве $(X, \leq)$

```
Choose mode:
Press 1 to get factor-set
Press 2 to create Hasse's diagramm
Press 3 to create a lattice of concepts
Press 4 to exit
3
Enter set of objects:
1 2 3 4
Enter set of attributes:
a b c d
Enter a binary relation matrix:
1 0 1 0
1 1 0 0
0 1 0 1
0 1 0 1
Closure system: [['1', '2', '3', '4'], ['1', '2'], ['2', '3', '4'], ['2'], ['1'], [], ['3', '4']]
Lattice of concepts: [[['1', '2', '3', '4'], []], [['1', '2'], ['a']], [['2', '3', '4'], ['b']], [['2'], ['a', 'b']],
[['1'], ['a', 'c']], [[], ['a', 'b', 'c', 'd']], [['3', '4'], ['b', 'd']]]
Choose mode:
Press 1 to get factor-set
Press 2 to create Hasse's diagramm
Press 3 to create a lattice of concepts
Press 4 to exit
```

Рисунок 4 – Тест алгоритма построения решетки концепта

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы были рассмотрены теоретические сведения об отношении эквивалентности, разобраны определения фактор-множества, отношения порядка и диаграммы Хассе, контекста и концепта. Опираясь на изложенную выше теорию, были разработаны алгоритмы построения эквивалентного замыкания бинарного отношения и системы представителей фактор-множества, алгоритмы вычисления минимальных (максимальных) и наименьших (наибольших) элементов и построения диаграммы Хассе, а также алгоритмы построения решетки концептов. Была произведена оценка сложности каждого из построенных алгоритмов. Была реализована программа, написанная на языке Python.