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

ОТНОШЕНИЕ ЭКВИВАЛЕНТНОСТИ И ОТНОШЕНИЕ ПОРЯДКА

ЛАБОРАТОРНАЯ РАБОТА

студента 3 курса 331 группы
специальности 100501 — Компьютерная безопасность
факультета КНиИТ
Окунькова Сергея Викторовича

Проверил
аспирант _____ В. Н. Кутин

Саратов 2022

СОДЕРЖАНИЕ

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.

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.

Порядок $\leq$ на множестве A называется:

1. линейным, если любые два элемента этого множества сравнимы, т.е. выполняется $(\forall a, b \in A)(a \leq b \vee b \leq a)$;

2. полным, если его любое непустое подмножество имеет точную верхнюю и точную нижнюю грани;
3. решеточным, если для любых $\forall a, b \in A$ существуют supa,b и infa,b, которые обозначаются соответственно $a \vee b$, $a \wedge b$ и называются также объединением и пересечением элементов a, b.

Множество с заданным на нем линейным порядком называется линейно упорядоченным множеством или цепью.

Множество с заданным на нем решеточным порядком называется решеточно упорядоченным множеством или решеткой.

Очевидно, что любая цепь является решеткой, но обратное в общем случае не выполняется.

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.

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

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:

$$k_{g,m} = \begin{cases} 1, & \text{если } (g, m) \in \rho \\ 0, & \text{если } (g, m) \notin \rho \end{cases}$$

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).

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 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

1. Алгоритм 1.1 - Замыкание бинарного отношения относительно рефлексивности:

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

   Выход: матрица бинарного отношения A', замкнутая относительно рефлексивности

   Шаг 1. Присвоить каждому $a[i][i]$ значение 1, где $0 \leq i < n$: , после чего вернуть полученную матрицу бинарного отношения $A' = (a'_{ij})$ с построенным на нем рефлексивным замыканием.

   Асимптотика $O(n)$.

2. Алгоритм 1.2 - Замыкание бинарного отношения относительно симметричности:

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

   Выход: матрица бинарного отношения A', замкнутая относительно симметричности

   Шаг1. Каждому элементу $a[i][j]$ $0 \leq i, j < n$ матрицы A присваивается значение элемента $a[j][i]$, после чего вернуть полученную матрицу бинарного отношения $A' = (a'_{ij})$ с построенным на нем симметричным замыканием.

   Асимптотика $O(n^2)$.

3. Алгоритм 1.3 - Замыкание бинарного отношения относительно транзитивности:

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

   Выход: матрица бинарного отношения A', замкнутая относительно транзитивности

   Шаг1. Если $a[i][k] = 1$ и $a[k][j] = 1$, то присвоить $a[i][j]$ значение 1, где $0 \leq i, j, k < k$. Такой шаг нужно повторить n раз в силу определения п.3) оператора транзитивного замыкания в лемме 2, после чего вернуть полученную матрицу бинарного отношения $A' = (a'_{ij})$ с построенным на нем транзитивным замыканием.

   Асимптотика $O(n^4)$.

4. Алгоритм 1.4 - Построение эквивалентного замыкания:

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

Выход: Матрица бинарного отношения $A''' = (a'''_{ij})$ с построенным на нем эквивалентным замыканием.

Шаг 1. Построить рефлексивное замыкание на бинарном отношении с матрицей $A = (a_{ij})$ с помощью алгоритма 1.1. Полученную матрицу бинарного отношения обозначить как $A' = (a'_{ij})$.

Шаг 2. Построить симметричное замыкание на бинарном отношении с матрицей $A' = (a_{ij})$ с помощью алгоритма 1.2. Полученную матрицу бинарного отношения обозначить как $A'' = (a''_{ij})$.

Шаг 3. Построить транзитивное замыкание на бинарном отношении с матрицей $A'' = (a_{ij})$ с помощью алгоритма 1.3. Полученную матрицу бинарного отношения обозначить как $A''' = (a''''_{ij})$.

Согласно пункту 4 леммы 1 о замыканиях бинарных отношений, построенное замыкание на данном бинарном отношении, определяемым матрицей, является эквивалентным.

Сложность алгоритма $O(n^4)$.

5. Алгоритм 2 - Построение системы представителей фактор-множества

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

   Выход: Система представителей T фактор-множества A/ε бинарного отношения на множестве A.

   Шаг 1. Получить фактор-множество A/ε бинарного отношения для множества A. Для этого нужно получить классы эквивалентности $\varepsilon(a)$, которые являются срезами по элементам множества A. Срезом по каждому элементу $a \in A$ является совокупность таких элементов множества A, значения которых в строке матрицы, определяющей связи между элементом a и другими элементами множества A, равны единице. Для этого проверим элементы a_{ij} матрицы A, и если $a_{ij} = 1$, где $0 \leq i, j \leq n-1$, то добавить значение j в список, определяющий срез по элементу i. В результате полученная совокупность всех таких срезов, являющихся классами эквивалентности $\{[a] : a \in A\}$, будет определять фактор-множество A/ε бинарного отношения A.

   Шаг 2. Отсортировать фактор-множество по возрастанию количества элементов в классах эквивалентности.

   Шаг 3. Проходясь по каждому элементу a каждого класса эквивалентно-

сти $[a]$ проверять: если элемент a класса эквивалентности не находится в системе представителей - добавить элемент в систему представителей как представителя класса эквивалентности $[a]$, иначе - пропустить элемент. Затем вернуть полученную систему представителей.
Сложность алгоритма $O(n^2)$.

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

1. Алгоритм 3 - Вычисление минимального элемента множества
   Вход: Матрица бинарного отношения $A = (a_{ij})$ размерности $n \times n$ и упорядоченное множество элементов S размерности n, относительно которого была построенна матрица A.
   Выход: Список минимальный элемент упорядоченного множества S.
   Шаг 1. Получить список срезов B матрицы A: если $a_{ij} = 1$, где $0 \leq i, j \leq n-1$, то добавить значение j в список, определяющий срез по элементу i.
   Шаг 2. Пройтись по каждому элементу множества B и найти такой элемент b_i, где $0 \leq i < n$, чтобы длина b_i была $\leq$ длине любого другого элемента множества B, затем длину этого элемента присвоить переменной l.
   Шаг 3. Создать пустой список C. Пройтись по каждому элементу множества B и, если длина $b_i = l$, где $0 \leq i < n$, то добавить соответствующий срезу b_i элемент s_i. Затем вернуть полученный список C.
   Сложность алгоритма $O(n^2)$.
2. Алгоритм 4 - Вычисление наименьшего элемента множества
   Вход: Матрица бинарного отношения $A = (a_{ij})$ размерности $n \times n$ и упорядоченное множество элементов S размерности n, относительно которого была построенна матрица A.
   Выход: Наименьший элемент упорядоченного множества S или None, если такого элемента не существует.
   Шаг 1. С помощью алгоритма 3 получить список минимальных элементов упорядоченного множества S. Если в списке находится один элемент, то вернуть его, иначе вернуть None.
   Сложность алгоритма $O(n^2)$.
3. Алгоритм 5 - Вычисление максимального элемента множества
   Вход: Матрица бинарного отношения $A = (a_{ij})$ размерности $n \times n$ и упорядоченное множество элементов S размерности n, относительно которого

была построенна матрица A.

Выход: Список максимальных элемент упорядоченного множества S.

Шаг 1. Получить список срезов B матрицы A: если $a_{ij} = 1$, где $0 \leq i, j \leq n-1$, то добавить значение j в список, определяющий срез по элементу i.

Шаг 2. Пройтись по каждому элементу множества B и найти такой элемент b_i, где $0 \leq i < n$, чтобы длина b_i была $\geq$ длине любого другого элемента множества B, затем длину этого элемента присвоить переменной l.

Шаг 3. Создать пустой список C. Пройтись по каждому элементу множества B и, если длина $b_i = l$, где $0 \leq i < n$, то добавить соответствующий срезу b_i элемент s_i. Затем вернуть полученный список C.

Сложность алгоритма $O(n^2)$.

4. Алгоритм 6 - Вычисление наибольшего элемента множества

   Вход: Матрица бинарного отношения $A = (a_{ij})$ размерности $n \times n$ и упорядоченное множество элементов S размерности n, относительно которого была построенна матрица A.

   Выход: Наибольший элемент упорядоченного множества S или None, если такого элемента не существует.

   Шаг 1. С помощью алгоритма 5 получить список максимальных элементов упорядоченного множества S. Если в списке находится один элемент, то вернуть его, иначе вернуть None.

   Сложность алгоритма $O(n^2)$.

5. Алгоритм 7 - Построение диаграмы Хассе для отношения делимости

   Вход: Матрица бинарного отношения делимости $A = (a_{ij})$ размерности $n \times n$ и упорядоченное множество элементов $S = (s_i)$ размерности n.

   Выход: Список уровней C каждого элемента множества S.

   Шаг 1. Создать спосок уровней C, в котором c_i будет соответствовать уровню s_i в диаграмме ($0 \leq i < n$), всем элементам присвоить уровень 0. И создать переменную lvl, в которой будет хранится следующий уровень диаграмы Хассе.

   Шаг 2. Найти с помощью алгоритма 3 список минимальных элементов множества S и присвоить соответствующим им элементам в списке C переменную lvl, после чего удалить соответствующие строки и столбцы в матрицк A и убрать эти элементы из множества C.

   Шаг 3. Если множество S стало пустым - вернуть C, иначе вернуться к

шагу 2.

Сложность алгоритма $O(n^3)$.

3.3 Описание алгоритма построения решетки концептов

Алгоритм 8 - Построение решетки концептов.

Вход: Матрица бинарного отношения $A = (a_{ij})$ размерности $n \times k$, множеством объектов G размерности n и множеств атрибутов M размерности k.

Выход: Решетка концептов C.

Шаг 1. Определить пустой список C.

Шаг 2. Положим, что $Z_{f_G} = A$.

Шаг 3. Для каждого элемента m_i матрицы M, где $0 \leq i < k$ вычисим его срез в матрице A, после чего добавим всего его не пустые пересечения со срезами матрицы Z_{f_G} по атрибутам вместе с новым значением атрибута, которое будет вычислятся как обединение атрибутов, по которым брались срез в матрице A и Z_{f_G}, в список C.

Шаг 4. Положим, что $Z_{f_G} = C$.

Шаг 3. Проделать шаг 2 и 3 n раз. После этого вернуть построенную решетку концептов $(C(K), \leq)$.

Трудоемкость алгоритма $O(k^2 \cdot n^2)$.

3.4 Коды программ, реализующей рассмотренные алгоритмы

```
from re import I, M
import numpy as np
import matplotlib.pyplot as plt


def isTran(a):
    n = len(a)
    b = np.matmul(a, a)
    for i in range(n):
        for j in range(n):
            if b[i][j]:
                b[i][j] = b[i][j] / b[i][j]
    f = (a >= b).all()
    if f:
        print("Set is transitive")
        return 't'
    else:
```

```
        f1 = True
        for i in range(n):
            for j in range(n):
                for k in range(n):
                    if a[i][k] and a[k][j] and a[i][j]:
                        f1 = False
        if f1:
            print("Set is anti-transitive")
            return 'at'
        else:
            print("Set is not transitive")
            return 'nt'


def isSymm(a):
    b = a.transpose()
    if np.array_equal(a, b):
        print("Set is symmetry")
        return 's'
    else:
        f = True
        b = np.multiply(a, b)
        for i in range(len(b)):
            for j in range(len(b[i])):
                if b[i][j] != 0 and i != j:
                    f = False
                    break
        if f:
            print("Set is anti-symmetry")
            return 'as'
        else:
            print("Set is not symmetry")
            return 'ns'


def isRefl(a):
    n = len(a)
    sum = 0
    for i in range(n):
        sum += a[i][i]
    if sum == n:
```

```
        print("Set is reflexive")
        return 'r'
    elif sum == 0:
        print("Set is anti-reflexive")
        return 'ar'
    else:
        print("Set is not reflexive")
        return 'nr'


def makeRefl(a):
    n = len(a)
    for i in range(n):
        a[i][i] = 1


def makeSymm(a):
    n = len(a)
    for i in range(n):
        for j in range(n):
            if a[i][j]:
                a[j][i] = 1


def makeTran(a):
    n = len(a)
    for c in range(n):
        for k in range(n):
            for i in range(n):
                for j in range(n):
                    if a[i][k] and a[k][j]:
                        a[i][j] = 1

def get_set(a):
    s = []
    for i in range(len(a)):
        for j in range(len(a[i])):
            if a[i][j] == 1:
                s.append((i + 1, j + 1))
    return s
```

```
def get_slices_list(relation_matrix, for_factor_set=1, attributes=None):
    slices = []
    for i in range(len(relation_matrix)):
        some_slice = [].copy()
        for j in range(len(relation_matrix[i])):
            if relation_matrix[i][j]:
                if attributes is None:
                    some_slice.append(j + for_factor_set)
                else:
                    some_slice.append(attributes[j])
        some_slice = tuple(some_slice)
        slices.append(some_slice)
    return slices


def create_factor_set(bin_relation):
    slices = get_slices_list(bin_relation)
    print("Factor set:")
    factor_set = set(slices)
    print(factor_set)
    return factor_set


def create_representative_system(factor_set):
    representative_system = []
    factor_set = list(factor_set)
    factor_set.sort(key=len)

    elem_dict = {}

    for eq_class in factor_set:
        for representative in eq_class:
            if not(representative in representative_system):
                elem_dict[f"{representative}"] = eq_class
                representative_system.append(representative)
                break

    print("Representative set factor system:")
    print(representative_system)
    ans = ""
    print("Where:", ans)
```

```
    for k, v in elem_dict.items():
       print(f"{k} \u2208 {list(v)}; ")


def get_dividers(num, set = None):
    i = 1
    ans = []
    if set == None:
        while(num >= i):
            if num % i == 0:
                ans.append(i)
            i += 1
    else:
        for s in set:
            if num % s == 0:
                ans.append(s)
    return ans


def get_level(a, dct):
    for an in a:
        m = 0
        for h in an:
            if m < dct[h]:
                m = dct[h]
        dct[an[len(an) - 1]] = m + 1
    return dct


def visualization(levels, max_level):
    plt.xlim(-1.0, max_level * 2)
    plt.ylim(0, len(levels) * 3 + 1.5)
    lvl = 1.5
    a = []
    for i in range(len(levels)):
        x = (max_level - len(levels[i]))
        l = 0
        b = []
        for j in range(len(levels[i])):
            plt.text(x + l, i + lvl, f'{levels[i][j]}')
            plt.scatter(x + l + 0.1, i + lvl + 0.15, s=350, facecolors='none', edgeco
```

```
            b.append((x + l, i + lvl, levels[i][j]))
            if i > 0:
                div = []
                for g in range(len(a) - 1, -1, -1):
                    for c in a[g]:
                        flag = True
                        for d in div:
                            if d % c[2] == 0:
                                flag = False
                        if levels[i][j] % c[2] == 0 and flag:
                            plt.plot([c[0] + 0.15, x + l + 0.15], [c[1] + 0.65, i + lv
                            div.append(c[2])
            l += 2
        a.append(b)
        lvl += 2
    plt.show()


def get_max_elem(dct, st = [], is_matrix = False):
    if not(is_matrix):
        m = -10000000
        for key in dct:
            m = max(m, key)
        return m
    else:
        a = get_slices_list(dct)
        lst = [len(b) for b in a]
        l = max(lst)
        ans = []
        for i in range(len(lst)):
            if lst[i] == l:
                ans.append(st[i])
        return ans


def get_min_elem(dct, st = [], is_matrix = False):
    if not(is_matrix):
        m = 10000000
        for key in dct:
            m = min(m, key)
        return m
```

```
    else:
        a = get_slices_list(dct)
        print(a)
        lst = [len(b) for b in a]
        l = min(lst)
        ans = []
        for i in range(len(lst)):
            if lst[i] == l:
                ans.append(st[i])
        return ans


def get_smallest_element(dct, st =[], is_matrix = False):
    if not(is_matrix):
        m = 10000000
        for key in dct:
            m = min(m, key)
        cnt = 0
        for key in dct:
            if key == m:
                cnt += 1
        if cnt == 1:
            return m
        else:
            return None
    else:
        ans = get_min_elem(dct, st, is_matrix)
        if len(ans) == 1:
            return ans[0]
        else:
            return None


def get_largest_element(dct, st = [], is_matrix = False):
    if not(is_matrix):
        m = -1000000
        for key in dct:
            m = max(m, key)
        cnt = 0
        for key in dct:
            if key == m:
```

```
                cnt += 1
        if cnt == 1:
            return m
        else:
            return None
    else:
        ans = get_max_elem(dct, st, is_matrix)
        if len(ans) == 1:
            return ans[0]
        else:
            return None


def task1():
    print("How you want enter your relation (set or matrix)?")
    enter = input()
    if enter == 'set':
        print('Enter the number of elements in relation')
        n = int(input())
        print("Enter your set")
        # st = [(1, 3), (3, 4), (1, 4), (2, 5), (5, 3)]
        s = list(map(str, input().split(' ')))
        st = []
        for c in s:
            a = ''
            i = 1
            while c[i] != ',':
                a += c[i]
                i += 1
            i += 1
            b = ''
            while c[i] != ')':
                b += c[i]
                i += 1
            st.append((int(a), int(b)))
        a = np.zeros((n, n), int)
        for s in st:
            a[s[0] - 1][s[1] - 1] = 1
        print('---------------------------------------')
        print("Relation's matrix")
        print(a)
```

```
    else:
        print('Enter the number of elements in relation')
        n = int(input())
        print("Enter your matrix")
        a = [list(map(int, input().split())) for i in range(n)]
        a = np.array(a).reshape(n, n)
        print('----------------------------------------')
        print("Relation's set")
        print(*get_set(a))
    print('----------------------------------------')
    properties = {"reflexive": isRefl(a), "symmetry": isSymm(a), "transitive": isTran
    print('----------------------------------------')
    if properties['reflexive'] == 'r' and properties['symmetry'] == 's' and properties
        print('Relation is equivalent')
        print('----------------------------------------')
    if properties['reflexive'] == 'r' and properties['transitive'] == 't':
        print('Relation is the relation of the quasi-order')
        print('----------------------------------------')
    if properties['reflexive'] == 'r' and properties['symmetry'] == 'as' and properti
        print('Relation is the relation of the partial order')
        print('----------------------------------------')
    if properties['reflexive'] == 'ar' and properties['symmetry'] == 'as' and properti
        print('Relation is the relation of the strict order')
        print('----------------------------------------')
    if properties['reflexive'] != 'r':
        makeRefl(a)
    if properties['symmetry'] != 's':
        makeSymm(a)
    makeTran(a)
    print('Closure matrix: ')
    print(a)
    print('----------------------------------------')
    print('Closure set: ')
    print(*get_set(a))
    print('----------------------------------------')

    factor_set = create_factor_set(a)
    create_representative_system(factor_set)


def get_level_matrix(matrix, st):
```

```
    dct = {st[i]: 0 for i in range(len(st))}
    i = 1
    matrix = np.array(matrix, int)
    while(matrix.any()):
        m = get_min_elem(matrix, st=st, is_matrix=True)
        for elem in m:
            dct[elem] = i
            matrix = np.delete(matrix, [st.index(elem)], 0)
            matrix = np.delete(matrix, [st.index(elem)], 1)
            st.remove(elem)
        print(matrix)
        i += 1
    return dct


def task2():
    print("The relation of divisibility on a set or on a number or matrix")
    h = input()
    if h == "number":
        print('Enter your number')
        num = int(input())
        dividers = get_dividers(num)
        print(f'Dividers of {num} : {dividers}')
        print(f"Maximum element is {get_max_elem(dividers)}")
        print(f"Minimum element is {get_min_elem(dividers)}")
        print(f"The smallest element is {get_smallest_element(dividers)}")
        print(f"The largest element is {get_largest_element(dividers)}")
        a = [get_dividers(div) for div in dividers]
        print(a)
        dct = {dividers[i]: 0 for i in range(len(a))}
        dct = get_level(a, dct)
    elif h == "set":
        print('Enter your set')
        st = list(map(int, input().split()))
        print(f"Maximum element is {get_max_elem(st)}")
        print(f"Minimum element is {get_min_elem(st)}")
        ls = get_smallest_element(st)
        bs = get_largest_element(st)
        if ls:
            print(f"The smallest element is {ls}")
        else:
```

```
            print("The smallest element doesn't exist")
        if bs:
            print(f"The largest element is {bs}")
        else:
            print("The largest element doesn't exist")
        st = list(set(st))
        a = [get_dividers(div, set=st) for div in st]
        dct = {st[i]: 0 for i in range(len(a))}
        dct = get_level(a, dct)
    else:
        print('Enter your set')
        st = list(map(int, input().split()))
        st = list(set(st))
        print("Enter your matrix")
        print("  ", *st)
        matrix = []
        a = []
        for i in range(len(st)):
            print(st[i], end="  ")
            b = []
            s = list(map(int, input().split()))
            for i in range(len(s)):
                if s[i]:
                    b.append(st[i])
            a.append(b)
            matrix.append(s)
        print(f"Maximum element is {get_max_elem(matrix, st=st, is_matrix=True)}")
        print(f"Minimum element is {get_min_elem(matrix, st=st, is_matrix=True)}")
        ls = get_smallest_element(matrix, st=st, is_matrix=True)
        bs = get_largest_element(matrix, st=st, is_matrix=True)
        if ls:
            print(f"The smallest element is {ls}")
        else:
            print("The smallest element doesn't exist")
        if bs:
            print(f"The largest element is {bs}")
        else:
            print("The largest element doesn't exist")
        dct = get_level_matrix(matrix, st)
    last_level = 1
    for key in dct:
```

```
            last_level = max(dct[key], last_level)
        print(dct)
        levels = [[] for _ in range(last_level)]
        for key in dct:
            levels[dct[key] - 1].append(key)
        max_level = 1
        for lvl in levels:
            max_level = max(max_level, len(lvl))
        visualization(levels, max_level)


def task3():
    print("Enter your set:")
    st = list(map(int, input().split()))
    print("Enter your context:")
    context = list(input().split())
    print("Enter your matrix")
    print("  ", *context)
    matrix = []
    for i in range(len(st)):
        print(st[i], end="  ")
        s = list(map(int, input().split()))
        matrix.append(s)
    matrix = np.array(matrix)
    dct = {}
    for i in range(len(context)):
        dct[tuple(context[i])] = matrix[:, i]
    dct1 = {}
    dct2 = dct
    for i in range(len(st)):
        for key in dct:
            for k in dct2:
                a = np.multiply(dct2[k], dct[key])
                cnt = 0
                for c in a:
                    cnt += c
                if cnt and key != k:
                    g = list(key)
                    g.extend(list(k))
                    g.sort()
                    dct1[tuple(set(g))] = a
```

```
                elif cnt:
                    dct1[tuple(key)] = a
        dct2 = dct1.copy()
    dl = []
    for key in dct1:
        for k in dct1:
            if np.array_equal(dct1[key],dct1[k]):
                if len(key) > len(k):
                    dl.append(k)
                elif len(key) < len(k):
                    dl.append(key)
    dl = list(set(dl))
    for d in dl:
        del dct1[d]
    for key in dct1:
        b = dct1[key]
        a = []
        for i in range(len(b)):
            if b[i]:
                a.append(st[i])
            dct1[key] = a
    print(dct1)


if __name__ == "__main__":
    print("What are you want? (1 - Create factor set, 2 - Build Hasse diagram, 3 - bu
    f = int(input())
    if f == 1:
        task1()
    elif f == 2:
        task2()
    elif f == 3:
        task3()
    else:
        print("Something going wrong! Enter a number from 1 to 3")
```

3.5 Результаты тестирования программ

```
What are you want? (1 - Create factor set, 2 - Build Passe diagram, 3 - build lattice concept)
1
How you want enter your relation (set or matrix)?
matrix
Enter the number of elements in relation
4
Enter your matrix
1 0 1 0
0 1 0 1
0 0 1 0
0 1 0 1
----------------------------------------
Relation's set
(1, 1) (1, 3) (2, 2) (2, 4) (3, 3) (4, 2) (4, 4)
----------------------------------------
Set is reflexive
Set is not symmetry
Set is transitive
----------------------------------------
Relation is the relation of the quasi-order
----------------------------------------
Closure matrix:
[[1 0 1 0]
 [0 1 0 1]
 [1 0 1 0]
 [0 1 0 1]]
----------------------------------------
Closure set:
(1, 1) (1, 3) (2, 2) (2, 4) (3, 1) (3, 3) (4, 2) (4, 4)
----------------------------------------
Factor set:
{(2, 4), (1, 3)}
Representative set factor system:
[2, 1]
Where:
2 ∈ [2, 4];
1 ∈ [1, 3];
```

Рисунок 1 – Тест алгоритма построения фактор множества

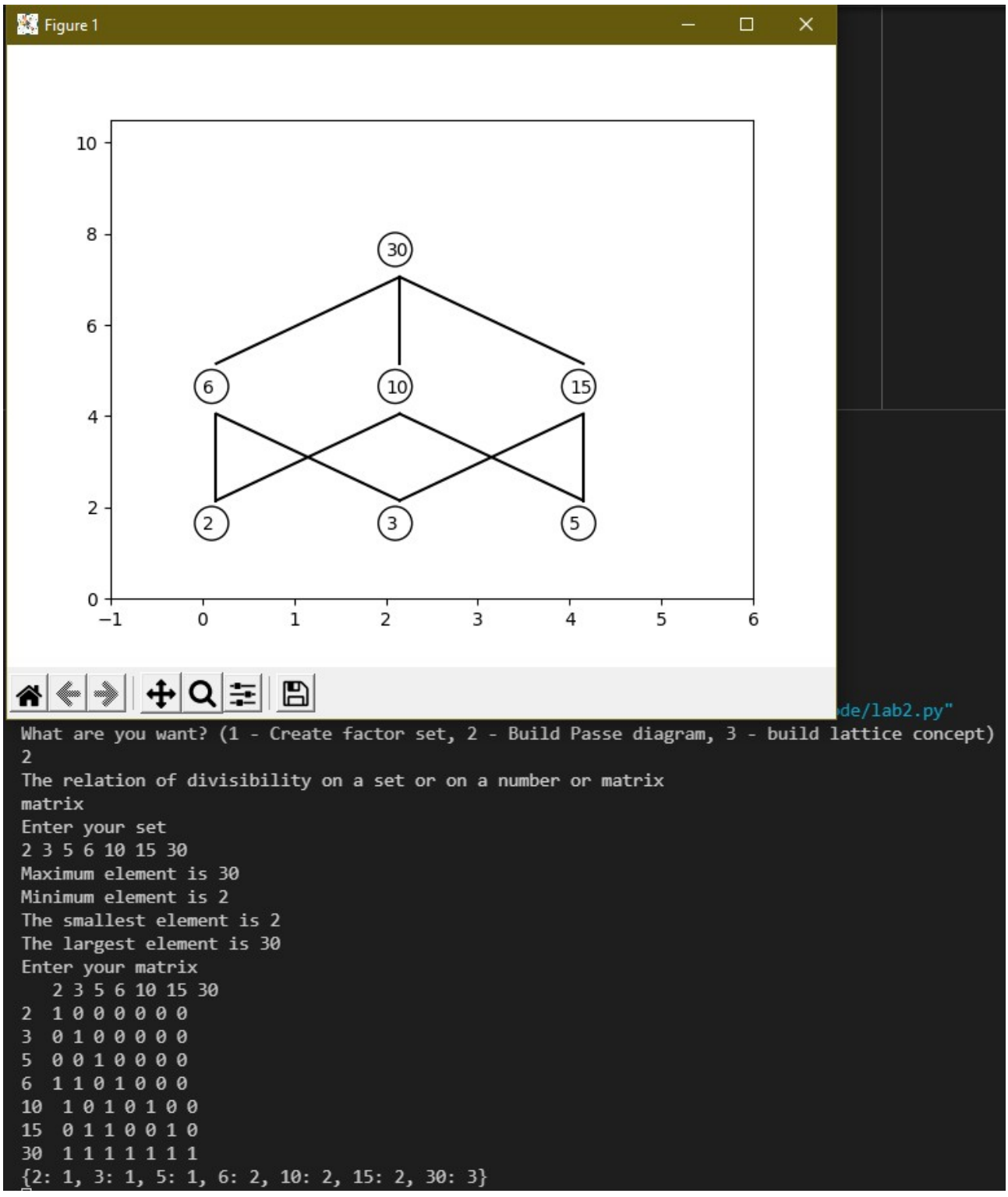

Рисунок 2 – Тест алгоритма построения диаграммы Хассе и поиска максимального(минимального) и наибольшего(наименьшего) элемента множества

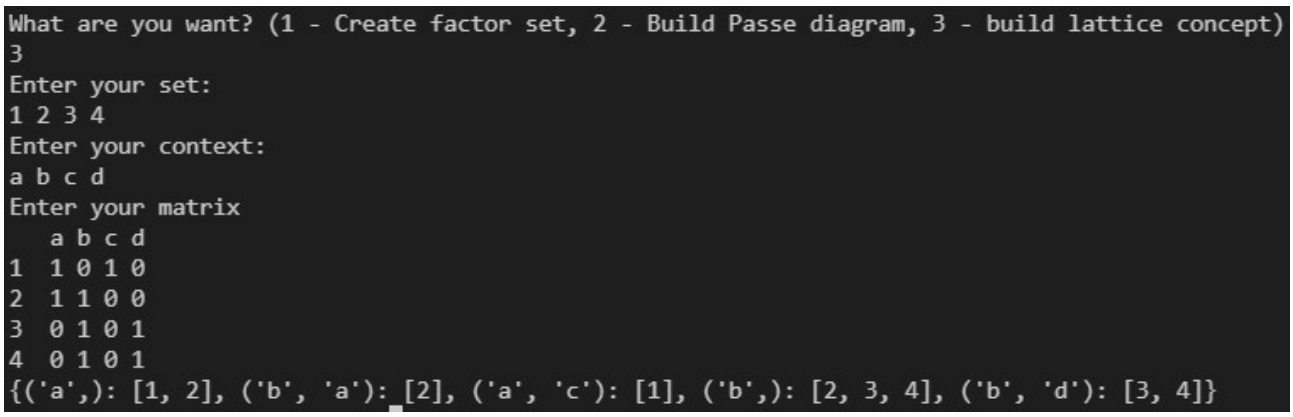

Рисунок 3 – Тест алгоритма построения решетки концептов

3.6 Оценки сложности рассмотренных алгоритмов

3.6.1 Алгоритм построения эквивалентного замыкания бинарного отношения

С учетом результатов вычисления асимптотики для первой лабораторной работы, алгоритм эквивалентного замыкания имеет асимптотику $O(n^4)$.

3.6.2 Алгоритм системы представителей.

С учетом наличия одного вложенного цикла и применения быстрой сортировки, алгоритм системы представителей имеет асимптотику $O(n^2 + nlogn) = O(n^2)$.

3.6.3 Алгоритм нахождения минимальных (максимальных) элементов множества.

Временная сложность получения срезов составляет $O(n^2)$, сложность нахождение минимальных (максимальных) элементов по длине среза составляет $O(n)$. Исходя из всего этого временная сложность алгоритма составляет $O(n^2)$.

3.6.4 Алгоритм нахождения наименьших (наибольших) элементов множества.

Если не учитывать асимптотику алгоритма нахождения минимальных (максимальных) элементов множества, который используется в этом алгоритме, то сложность составляет $O(1)$, с учетом вышеописанного алгоритма асимптотика определяется как $O(n^2)$.

3.6.5 Алгоритм построения диаграммы Хассе.

С учетом наличия двух вложенных циклов, алгоритм построения диаграммы Хассе имеет асимптотику $O(n^3)$.

3.6.6 Алгоритм построения решетки концепта.

Асимптотика получения среза по столбцу атрибутов составляет $O(n^2)$, а эти срезы мы получаем k^2 раз. Следовательно алгоритм построения решетки концептов имеет асимптотику $O(k^2 \cdot n^3)$.

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы были рассмотренны теоритические основы отношений эквивалентности, построение их фактор множеств, построение диаграмм Хассе и решетки концептов. На основе этой теоретической части была смоделирована программа на языке Python с использованием средств библиотеки Numpy, которая способна пострить замыкание эквивалентности, фактор множество, систему представителей этого множества, построить диаграмму Хассе на отношение делимости и построить решетку концептов для заданного множества, а так же была оценена асимптотика каждого реализованного алгоритма.