МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Отношение эквивалентности и отношение порядка

ОТЧЁТ
ПО ДИСЦИПЛИНЕ
«ПРИКЛАДНАЯ УНИВЕРСАЛЬНАЯ АЛГЕБРА»

студента 3 курса 331 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Никитина Арсения Владимировича

Преподаватель
профессор, д.ф.-м.н. _____ В. А. Молчанов
подпись, дата

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Бинарные отношения могут быть эквивалентными, и, поэтому на них могут строиться фактор-множества. Если же бинарное отношение не является эквивалентностью, то по определенному алгоритму можно построить эквивалентное замыкание данного отношения. Также отношения могут обладать определенным порядком, в зависимости от конкретных свойств. Если же отношение обладает порядком, то для данного отношения можно построить диаграмму Хассе, а также для него могут быть найдены минимальные и максимальные, и наименьшие и наибольшие элементы. Также для бинарных отношений определены понятия контекста и концепта, а также существует алгоритм вычисления решетки концептов.

1 Цель работы и порядок ее 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 Теоретические сведения

2.1 Эквивалентное замыкание бинарного отношения

2.1.1 Системы замыканий бинарных отношений

Множество Z подмножеств множества A называется **системой замыканий**, если оно замкнуто относительно пересечений, т.е. выполняется:

$$\cap B \in Z \text{ для любого подмножества } B \subset Z$$

Лемма о системах замыканий бинарных отношений. На множестве $P(A^2)$ всех бинарных отношений между элементами множества A следующие множества являются системами замыканий:

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

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

2.1.2 Замыкания бинарных отношений

Итак, существуют 4 вида замыканий отношений: **транзитивное, симметричное, рефлексивное и эквивалентное**.

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

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

2.1.3 Алгоритм построения эквивалентного замыкания бинарного отношения

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

Выход. Эквивалентное замыкание бинарного отношения.

1. Создать пустой список для хранения пар замыкания.
2. Цикл по u от 1 до N.
   - *а*) Если $M_{uu} = 0$, пару (u, u) добавить в замыкание, M_{uu} присвоить значение 1.
   - *б*) Цикл по k от 1 до N.
     - i. Если $M_{uk} = 1$ и $M_{ku} = 0$, пару (k, u) добавить в замыкание, M_{ku} присвоить значение 1.
     - ii. Цикл по i от 1 до N, цикл по j от 1 до N.
       1. Если $M_{ki} = M_{ij} = 1$ и $M_{kj} = 0$, пару (k, j) добавить в замыкание, M_{kj} присвоить значение 1.
3. Ответ — эквивалентное замыкание бинарного отношения ρ.

Трудоемкость алгоритма $O(N^4)$.

2.2 Фактор-множество отношения

2.2.1 Определение среза отношения через элемент

Для любого подмножества $X \subset A$ множество:

$$\rho(X) = \{b \in B : (x, b) \in \rho \text{ для некоторого } X\}$$

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.

2.2.2 Определение фактор-множества отношения

Эквивалентное бинарное отношение на множестве A также принято обозначать как ε.

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 A/ε.

2.2.3 Алгоритм построения фактор-множества бинарного отношения

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

Выход. Фактор-множество отношения.

1. Создать пустое множество S.
   a) Цикл по i от 1 до N.
      i. Создать пустое множество S_1.
      ii. Цикл по j от 1 до N.
         1. Если $M_{ij} = 1$, добавить j во множество S_1.
      iii. Добавить получившееся множество S_1 в S.
2. Ответ — фактор-множество S отношения ρ.

Трудоемкость алгоритма $O(N^2)$.

2.3 Полная система представителей классов эквивалентности

2.3.1 Определение полной системы представителей классов эквивалентности

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.

2.3.2 Алгоритм получения полной системы представителей классов эквивалентности

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

Выход. Полная система представителей классов эквивалентности.

1. Создать пустой список.
2. Запустить алгоритм получения фактор-множества S отношения ρ.
3. Цикл по i от 1 до $|S|$.
   a) Добавить минимальный элемент i-го множества из фактор-множества в список.

4. Ответ — полная система представителей классов эквивалентности.

Трудоемкость алгоритма – $O(N^3)$.

2.4 Отношение порядка и упорядоченное множество

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 $a \leq b$.

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

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

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

2.4.1 Определение упорядоченного множества

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

2.4.2 Определение минимальных (наименьших) и максимальных (наибольших) элементов упорядоченного множества

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

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

2.4.3 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.

2.4.4 Алгоритм построения диаграммы Хассе конечного упорядоченного множества

Теоретический алгоритм

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.

Псевдо-код алгоритма для множества с операцией деления

Вход: Упорядоченное множество A мощности N.

Выход: Список H длиной n, характеризующий диаграмму Хассе: каждый элемент в списке представляет собой три значения: элемент $a \in A$, значение его уровня l на диаграмме, список D элементов множества A, находящихся на уровне $l - 1$ и связанных с элементом a.

1. Создать пустой список H.
2. Создать словарь с ключами из элементов множества A и значениями, равными 1.
3. Цикл по i от 2 до N.
   а) Цикл по j от 1 до i.
      i. Если A_i делится на A_j, то элементу словаря с ключом A_i присвоить значение элемента словаря с ключом A_j + 1.
4. Цикл по key, $value$ из словаря.
   а) Создать пустой список Q.
   б) Цикл по $key1$, $value1$ из словаря.
      i. Если $value1 + 1 = value$ и key делится на $key1$, то добавить $key1$ в Q.
   в) Добавить кортеж $(key, value, Q)$ в список H.
5. Ответ — список, состоящий из элементов диаграммы Хассе, с уровнями и связями с предыдущими уровнями диаграммы.

Трудоемкость алгоритма — $O(|A| \cdot |A|) = O(|A|^2)$, где A - упорядоченное множество.

Алгоритм получения делителей числа

Вход: Число a.

Выход: Список делителей числа a.

1. Создать пустой список.
2. Цикл по i от 1 до $a/2 + 1$.
   a) Если a делится на i, то добавить i в список.
3. Ответ — список делителей числа a.

Трудоемкость алгоритма — $O(a/2 + 1)$.

Псевдо-код алгоритма для множества, задаваемого числом с операцией деления

Вход: Число z.

Выход: Список H длиной n, характеризующий диаграмму Хассе: каждый элемент в списке представляет собой три значения: элемент $a \in A$, значение его уровня l на диаграмме, список D элементов множества A, находящихся на уровне $l - 1$ и связанных с элементом a.

1. Вызвать алгоритм получения делителей числа от z и сохранить результат в список.
2. Вызвать алгоритм получения элементов диаграммы Хассе для множества с операцией деления.
3. Ответ — список, состоящий из элементов диаграммы Хассе, с уровнями и связями с предыдущими уровнями диаграммы.

Трудоемкость алгоритма — $O(|A| \cdot |A|) = O(|A|^2)$, где A – множество всех делителей числа z.

2.4.5 Алгоритм получения минимальных элементов упорядоченного множества

Вход: Упорядоченное множество A размерности N.

Выход. Минимальные элементы множества A.

1. Создать пустой список R и добавить в него первый элемент кортежа первого элемента множества.

2. Создать переменную m_l и присвоить ей значение второго элемента кортежа первого элемента множества.
3. Цикл по i от 2 до N.
   а) Если второй элемент кортежа A_i равен m_l, то добавить первый элемент кортежа A_i в R.
   б) Если второй элемент кортежа A_i не равен m_l, то выход из цикла.
4. Ответ — минимальные элементы множества A: R.

Трудоемкость алгоритма — $O(N)$.

2.4.6 Алгоритм получения наименьшего элемента упорядоченного множества

Вход: Упорядоченное множество A размерности N.

Выход. «Наименьшим элементом множества A является r» или «Наименьшего элемента в данном множестве нет».

1. Создать переменную r.
2. Вызвать алгоритм получения элементов диаграммы Хассе для множества с операцией деления.
3. Если вторые элементы кортежей (отвечающие за уровень элемента) первого и второго элемента полученного множества равны, то ответ — «Наименьшего элемента в данном множестве нет».
4. Если вторые элементы кортежей первого и второго элемента полученного множества различны, то r присвоить значение первого элемента первого кортежа. Ответ — «Наименьшим элементом множества A является r».

Трудоемкость алгоритма — $O(1)$.

2.4.7 Алгоритм получения максимальных элементов упорядоченного множества

Вход: Упорядоченное множество A размерности N.

Выход. Максимальные элементы множества A.

1. Создать пустой список R и добавить в него первый элемент кортежа последнего элемента множества.
2. Создать переменную m_l и присвоить ей значение второго элемента кортежа последнего элемента множества.
3. Цикл по i от $N - 1$ до 1.

а) Если второй элемент кортежа A_i равен m_l, то добавить первый элемент кортежа A_i в R.

б) Если второй элемент кортежа A_i не равен m_l, то выход из цикла.

4. Ответ — максимальные элементы множества A: R.

Трудоемкость алгоритма — $O(N)$.

2.4.8 Алгоритм получения наибольшего элемента упорядоченного множества

Вход: Упорядоченное множество A размерности N.

Выход. «Наибольшим элементом множества A является r» или «Наибольшего элемента в данном множестве нет».

1. Создать переменную r.
2. Вызвать алгоритм получения элементов диаграммы Хассе для множества с операцией деления.
3. Если вторые элементы кортежей (отвечающие за уровень элемента) последнего и предпоследнего элемента полученного множества равны, то ответ — «Наибольшего элемента в данном множестве нет».
4. Если вторые элементы кортежей первого и второго элемента полученного множества различны, то r присвоить значение первого элемента первого кортежа. Ответ — «Набольшим элементом множества A является r».

Трудоемкость алгоритма — $O(1)$.

2.5 Контексты и решетки концептов

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.

2.5.1 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.

2.5.2 Алгоритм получения элементов решетки концептов

Вход: Матрица бинарного отношения $M(\rho)$ размерности $N \times N$, множество атрибутов A.

Выход. Решетка концептов.

1. Создать пустое множество c_s, множество a_s, заполненного числами от 1 до N, пустой словарь s_a.
2. Цикл по i от 1 до N.
   - *а*) Создать пустое множество n_s.
   - *б*) Цикл по j от 1 до N.
     - i. Если $M(\rho)_{ij} = 1$, добавить j в n_s.
   - *в*) Присвоить a_s пересечение a_s и n_s.
   - *г*) Если множество c_s пустое:
     - i. Добавить n_s в c_s.
     - ii. В словаре s_a по ключу $A[i]$ присвоить значение n_s.
   - *д*) Создать пустое множество s.

е) Цикл по элементам u из множества c_s.

i. Создать множество ss и присвоить ему пересечение u с n_s.

ii. Если множество ss непустое:

А. Цикл по $key, value$ в словаре s_a:

— Если $value = u$, в словаре s_a по ключу key, $A[i]$ присвоить значение ss, выход из цикла.

Б. Во множество s добавить множество ss.

ж) Цикл по элементам u из множества s.

i. Во множество c_s добавить множество u.

з) Если множество n_s не находится во множестве c_s, во множество c_s добавить множество n_s и в словаре s_a по ключу $A[i]$ присвоить значение n_s.

3. Создать переменную $s_g = \varnothing$.
4. Цикл по $key, value$ в словаре s_a:

а) Если $value = a_s$, $s_g = value$, выход из цикла.

5. Ответ — решетка концептов.

Трудоемкость алгоритма — $O(N^3 + N^2 + N^1) = O(N^3)$.

3 Программная реализация рассмотренных алгоритмов

3.1 Результаты тестирования программы

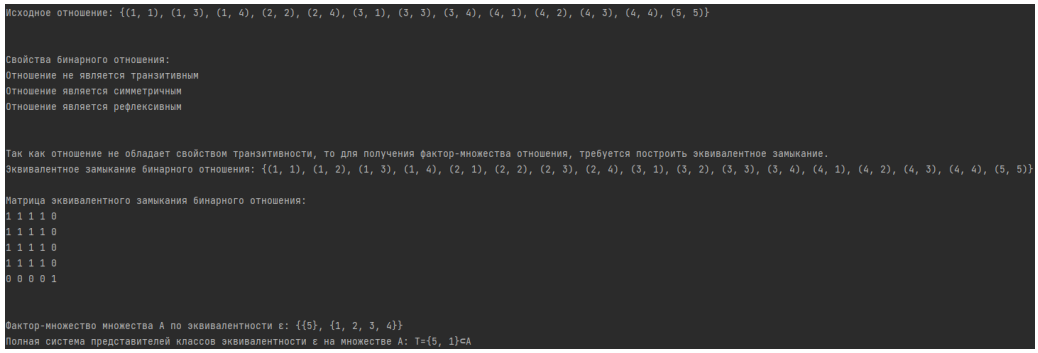

Рисунок 1

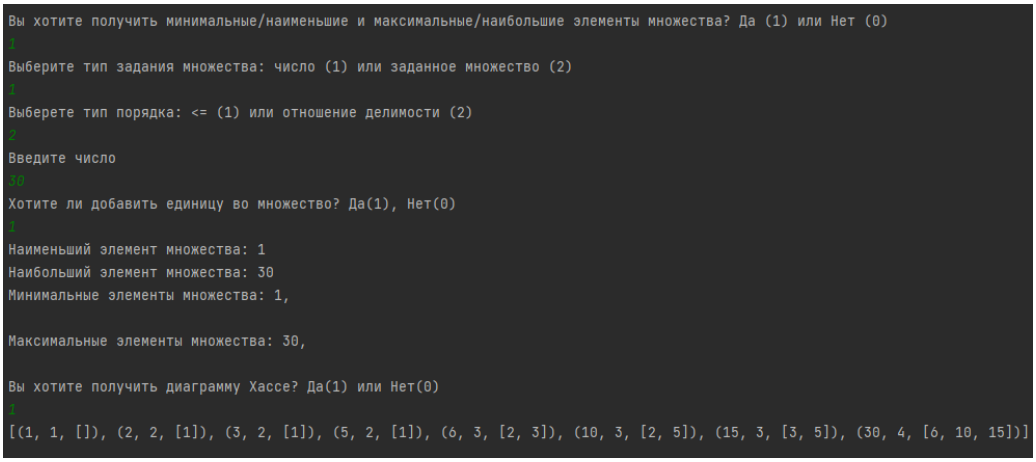

Рисунок 2

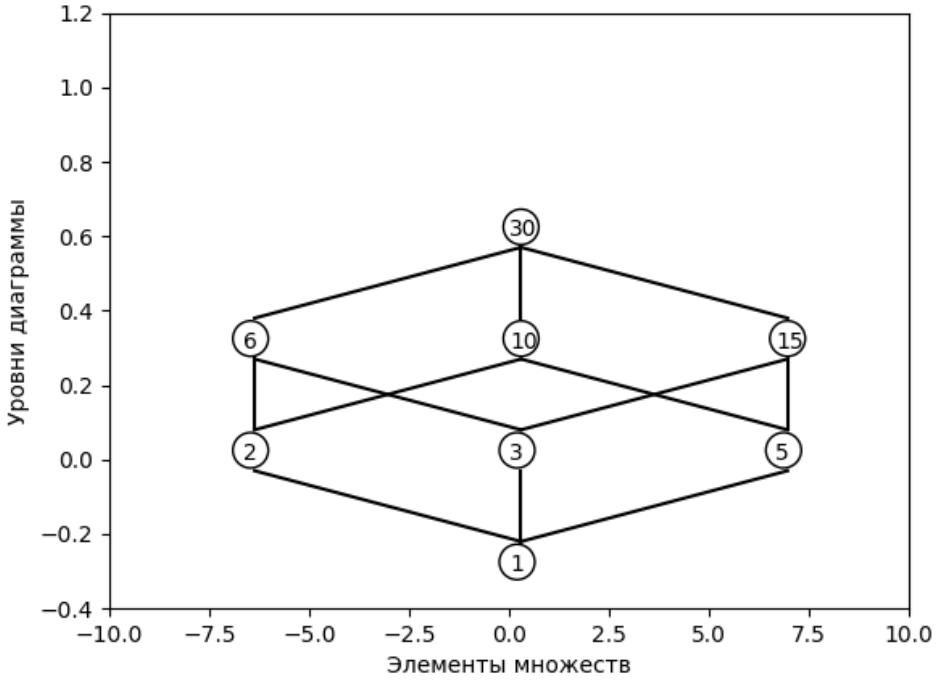

Рисунок 3

```
Вы хотите получить минимальные/наименьшие и максимальные/наибольшие элементы множества? Да (1) или Нет (0)
1
Выберите тип задания множества: число (1) или заданное множество (2)
2
Выберете тип порядка: <= (1) или отношение делимости (2)
2
Введите множество
2 3 21 15 14 4 8 30 16 32
Наименьшего элемента в данном множестве нет
Наибольший элемент множества: 32
Минимальные элементы множества: 2, 3,

Максимальные элементы множества: 32,

Вы хотите получить диаграмму Хассе? Да(1) или Нет(0)
1
```

Рисунок 4

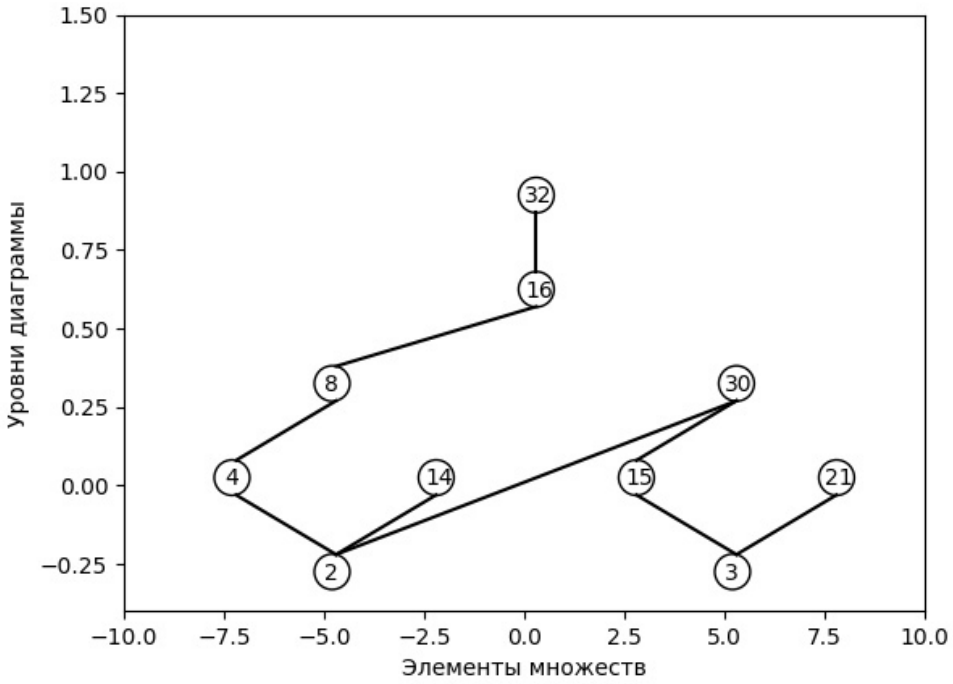

Рисунок 5

```
Вы хотите получить минимальные/наименьшие и максимальные/наибольшие элементы множества? Да (1) или Нет (0)
1

Выберите тип задания множества: число (1) или заданное множество (2)
1

Выберете тип порядка: <= (1) или отношение делимости (2)
1

Введите число
12
Хотите ли добавить единицу во множество? Да(1), Нет(0)
1

Наименьший элемент множества: 1
Наибольший элемент множества: 12
Минимальные элементы множества: 1,

Максимальные элементы множества: 12,

Вы хотите получить диаграмму Хассе? Да(1) или Нет(0)
1

[(1, 1, []), (2, 2, [1]), (3, 3, [2]), (4, 4, [3]), (5, 5, [4]), (6, 6, [5]), (7, 7, [6]), (8, 8, [7]), (9, 9, [8]), (10, 10, [9]), (11, 11, [10]), (12, 12, [11])]
```

Рисунок 6

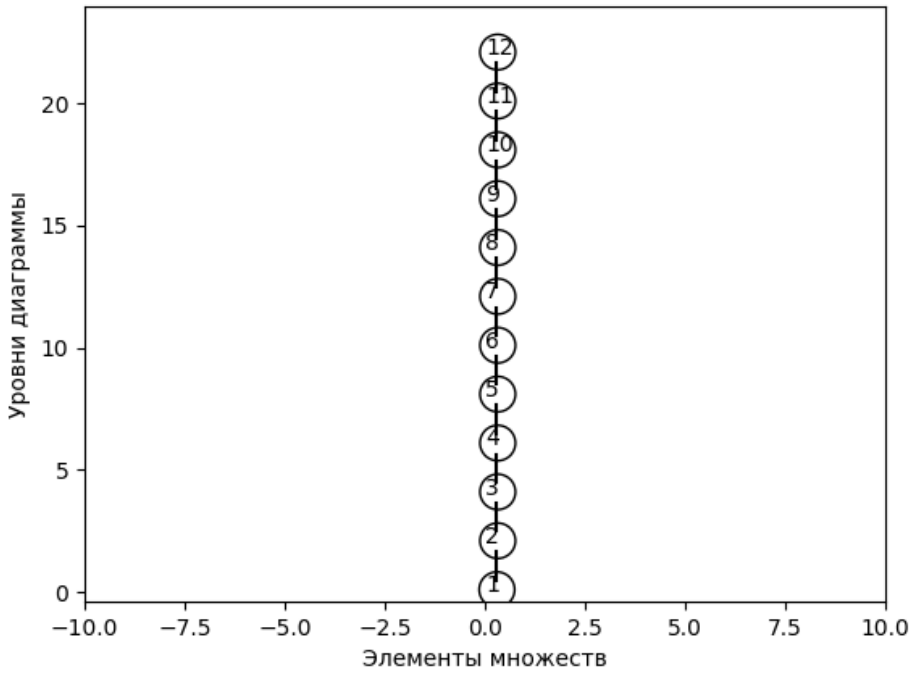

Рисунок 7

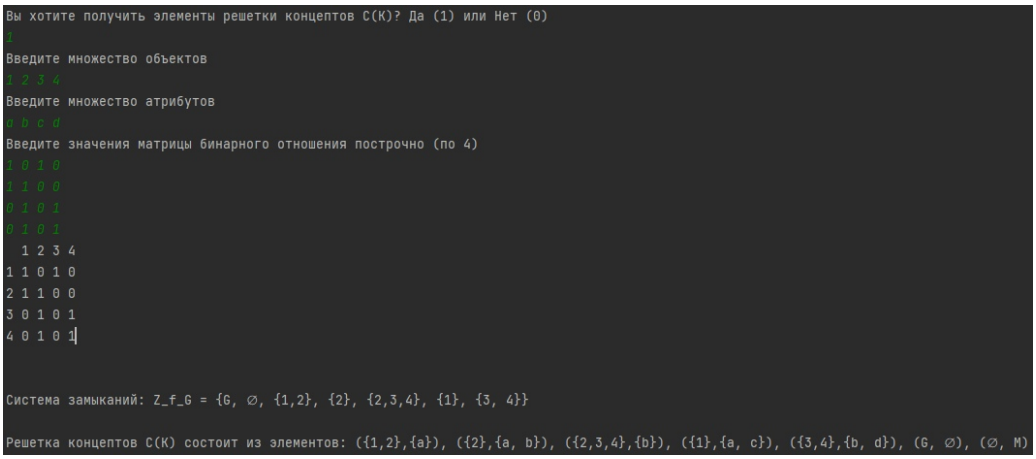

Рисунок 8

3.2 Коды программ, реализующих рассмотренные алгоритмы

3.2.1 Код программы, реализующей визуализацию диаграммы Хассе

```
1  import matplotlib.pyplot as plt
2
3
4  def dividers(num):
5      return [i for i in range(int(num / 2) + 1, 0, -1) if num % i == 0]
6
7
```

```
 8  def levels_length(lst):
 9      max_len = 1
10      max_level_length_list = []
11      value = lst[0][1]
12      for values in lst[1:]:
13          if values[1] == value:
14              max_len += 1
15          if values[1] != value:
16              max_level_length_list.append(max_len)
17              max_len = 1
18              value = values[1]
19
20      max_level_length_list.append(max_len)
21      return max_level_length_list
22
23
24  def get_levels_list(lst, len_levels):
25      levels_list = [[] for _ in range(len(len_levels))]
26      for value in lst:
27          levels_list[value[1] - 1].append(value[0])
28      return levels_list
29
30
31  def visual(lst, flag):
32
33      plt.xlim(-10.0, 10.0)
34      lim = lst[-1][1]
35      plt.xlabel('Элементы множестве')
36      plt.ylabel('Уровни диаграммы')
37
38      if flag == 1:
39          len_levels = levels_length(lst)
40
41          levels_numbers_list = get_levels_list(lst, len_levels)
42
43          plt.ylim(-0.4, 1.2 * lim / 4)
44
45          x_save = -10
46          y_value = -0.3
47          y_save = y_value
48
```

```
49          current_level = 0
50
51          dy = 0.3
52
53          for level in levels_numbers_list:
54              x_value = x_save
55              delta = 20 / len_levels[current_level]
56              delta1 = delta / 2
57              x_value += delta1
58              for value in level:
59                  plt.text(x_value, y_value, f'{value}')
60                  if value > 99:
61                      dx = 0.5
62                  elif value > 9:
63                      dx = 0.3
64                  else:
65                      dx = 0.2
66                  plt.scatter(x_value + dx, y_value + 0.025, s=250,
                    ↪ facecolors='none', edgecolors='black')
67                  x_value += delta
68              y_value += dy
69              current_level += 1
70
71          y_value = y_save + dy
72          for level, values in enumerate(levels_numbers_list[1:]):
73              level += 1
74              x_value = x_save
75              delta = 20 / len_levels[level]
76              delta1 = delta / 2
77              x_value += delta1
78              current_level = level
79              for value in values:
80                  dividers_lst = dividers(value)
81                  y_previous = dy
82                  for i, values_levels in enumerate(levels_numbers_list[level -
                    ↪ 1::-1]):
83                      x_value1 = x_save
84                      delta_value1 = 20 / len_levels[current_level - i - 1]
85                      delta1_value1 = delta_value1 / 2
86                      x_value1 += delta1_value1
87                      for value1 in values_levels:
```

```
88                          if value % value1 == 0 and value1 in dividers_lst:
89                              value1_dividers = dividers(value1)
90                              dividers_lst = [val for val in dividers_lst if val
                                ↪  not in value1_dividers + [value1]]
91                              plt.plot([x_value1 + 0.3, x_value + 0.3], [y_value
                                ↪  - y_previous + 0.08, y_value - 0.03],
92                                       color='black')
93                          x_value1 += delta_value1
94                      y_previous += dy
95                  x_value += delta
96              y_value += dy
97      elif flag == 2:
98          plt.ylim(-0.4, 2 * lim)
99
100         x_value = 0
101         y_value = 0
102         dy = 2 * lim / len(lst)
103
104         plt.text(x_value, y_value, f'{lst[0][0]}')
105         plt.scatter(x_value + 0.3, y_value + 0.1, s=250, facecolors='none',
            ↪  edgecolors='black')
106         y_value += dy
107         for value in lst[1:]:
108             plt.text(x_value, y_value, f'{value[0]}')
109             plt.scatter(x_value + 0.32, y_value + 0.1, s=250,
                ↪  facecolors='none', edgecolors='black')
110             plt.plot([x_value + 0.3, x_value + 0.3], [y_value - 0.35, y_value
                ↪  - dy + 0.5], color='black')
111             y_value += dy
112     elif flag == 3:
113         plt.xlim(-20.0, 20.0)
114         plt.ylim(-0.4, 1.2 * len(lst) / 4)
115         x_save = -20
116         y_value = -0.3
117         dy = 0.3
118         for level in lst:
119             x_value = x_save
120             delta = 30 / len(level)
121             delta1 = delta / 2
122             x_value += delta1
123             for value in level:
```

```
124                 plt.text(x_value, y_value, f'{value}')
125                 x_value += delta
126             y_value += dy
127
128     plt.show()
129
130
131 def get_levels_list_sets(sets):
132     res = [[] for _ in range(sets[-1][1])]
133     for subset in sets:
134         res[subset[1]-1].append(subset[0])
135     return res
136
137 ls = [(({2}, {'a', 'b'}), 1, []), (({1}, {'a', 'c'}), 1, []), (({3, 4},
    ↪ {'d', 'b'}), 1, []), (({1, 2}, {'a'}), 2, [{2}, {1}]), (({2, 3, 4},
    ↪ {'b'}), 2, [{2}, {3, 4}])]
138 print(get_levels_list_sets(ls))
139 visual(get_levels_list_sets(ls), 3)
140 '''
141 Примеры входных данных:
142
143 visual([(1, 1, []), (2, 2, [1]), (3, 3, [2]), (4, 4, [3]), (5, 5, [4]), (6, 6,
    ↪ [5]), (7, 7, [6]), (8, 8, [7]),
144         (9, 9, [8]), (10, 10, [9]), (11, 11, [10]), (12, 12, [11])], True)
145
146 visual([(1, 1, []), (2, 2, [1]), (3, 3, [2]), (4, 4, [3]), (5, 5, [4]), (6, 6,
    ↪ [5]), (7, 7, [6])], True)
147
148 visual([(1, 1, []), (2, 2, [1]), (3, 2, [1]), (5, 2, [1]), (6, 3, [2, 3]),
    ↪ (10, 3, [2, 5]), (15, 3, [3, 5]),
149          (30, 4, [6, 10, 15])])
150
151 [(('{a, b}', {2}), 1, []), (('{a, c}', {1}), 1, []), (('{b, d}', {3, 4}), 1,
    ↪ []), (('{a}', {1, 2}), 2, [{2}, {1}]), (('{b}', {2, 3, 4}), 2, [{2}, {3,
    ↪ 4}])]
152 '''
```

3.2.2 Код программы, реализующей получение решетки концептов

```
1 def make_nums_obj_attr(objects, attributes):
2     return {key: i for i, key in enumerate(objects)}, {key: i for i, key in
      ↪ enumerate(attributes)}
3
```

```
 4
 5  def get_lattice_of_concepts(matrix, size, keys):
 6      closure_system = set()
 7      subsets_attrs = dict()
 8      all_subsets = set([i + 1 for i in range(size)])
 9      for i in range(size):
10          new_subset = []
11          for j in range(size):
12              if matrix[j][i] == 1:
13                  new_subset.append(j + 1)
14          new_subset = frozenset(new_subset)
15          all_subsets = all_subsets.intersection(new_subset)
16          if not closure_system:
17              closure_system.add(new_subset)
18              subsets_attrs[keys[i]] = new_subset
19          else:
20              subsets = set()
21              for subset in closure_system:
22                  subsubset = frozenset(subset.intersection(new_subset))
23                  if subsubset:
24                      for key, value in subsets_attrs.items():
25                          if value == subset:
26                              subsets_attrs[f'{key}, {keys[i]}'] = subsubset
27                              break
28                      subsets.add(subsubset)
29              for subset in subsets:
30                  closure_system.add(subset)
31              if new_subset not in closure_system:
32                  closure_system.add(new_subset)
33                  subsets_attrs[f'{keys[i]}'] = new_subset
34      set_for_g = '\u2205'
35      for key, value in subsets_attrs.items():
36          if all_subsets == value:
37              set_for_g = value
38              break
39      return subsets_attrs, f'(G, {set_for_g})'
40
41
42  def get_matrix(size):
43      print(f'Введите значения матрицы бинарного отношения построчно (по
        ↪ {size})')
```

```
44      return [[int(value) for value in input().split()] for _ in range(size)]
45
46
47  def print_matrix(mat, obj):
48      print('  ', end='')
49      print(*obj)
50      symbols = list(obj.keys())
51      for i in range(len(mat)):
52          print(symbols[i], end=' ')
53          print(*mat[i])
54      print('\n')
55
56
57  def print_closure_system(dictionary):
58      print('Z_f_G = {G, \u2205, ', end='')
59      for value in dictionary[:-1:]:
60          value = list(value)
61          print('{', end='')
62          print(*value, sep=',', end='}, ')
63      print('{', end='')
64      print(*dictionary[-1], sep=', ', end='}')
65      print('}\n')
66
67
68  def intersections(sets):
69      result = []
70      for i, subset in enumerate(sets):
71          podres = [subset, 1, []]
72          for j, subset1 in enumerate(sets):
73              if not subset1[0].difference(subset[0]) and not subset1[0] ==
                ↪ subset[0]:
74                  podres[2].append(set(subset1[0]))
75          result.append(podres)
76
77      for subset in result:
78          for subset1 in result:
79              if not subset1[0][0].difference(subset[0][0]) and not
                ↪ subset1[0][0] == subset[0][0]:
80                  subset[1] = subset1[1] + 1
81
82      return sorted([tuple(val) for val in result], key=lambda x: x[1])
```

```
83
84
85  def print_lattice_of_concepts(mat, attr):
86      lattice_of_concepts, g = get_lattice_of_concepts(mat, len(attr),
        ↪ list(attr.keys()))
87      lattice_of_concepts_save = lattice_of_concepts.copy()
88      for key, value in lattice_of_concepts.items():
89          for key1, value1 in lattice_of_concepts.items():
90              if value == value1 and not key == key1:
91                  if min(key, key1) in lattice_of_concepts_save:
92                      lattice_of_concepts_save.pop(min(key, key1))
93      print('Система замыканий: ', end='')
94      print_closure_system(list(lattice_of_concepts_save.values()))
95      print('Включения-исключения))): ')
96      print(intersections([(set(v), set(k.replace(',', ' ').split())) for k, v
        ↪ in lattice_of_concepts_save.items()]))
97
98      # import hasse_visualization as hs
99      # hs.visual(res)
100     print('Решетка концептов C(K) состоит из элементов: ', end='')
101     for key, value in lattice_of_concepts_save.items():
102         value = list(value)
103         print('({', end='')
104         print(*value, sep=',', end='},')
105         print('{' + key + '}', end='), ')
106
107     print(g, end=', ')
108     print('(\u2205, M)')
109
110
111 def main():
112     print('Введите множество объектов')
113     # obj = [int(value) for value in input().split()]
114     obj = [1, 2, 3, 4]
115     print('Введите множество атрибутов')
116     # attr = input().split()
117     attr = ['a', 'b', 'c', 'd']
118     obj, attr = make_nums_obj_attr(obj, attr)
119
120     # mat = get_matrix(len(attr))
121     # mat = [[0, 1, 0, 1], [0, 1, 1, 0], [1, 0, 1, 0], [0, 0, 1, 1]]
```

```
122     mat = [[1, 0, 1, 0], [1, 1, 0, 0], [0, 1, 0, 1], [0, 1, 0, 1]]
123     print_matrix(mat, obj)
124     print_lattice_of_concepts(mat, attr)
125
126
127 main()
```

3.2.3 Код программы, реализующей основные алгоритмы

```
 1 def print_matrix_set(matrix_set, flag=None):
 2     if not flag:
 3         print('Исходное отношение: {', end='')
 4         print(*matrix_set, sep=', ', end='}\n')
 5     else:
 6         print('{', end='')
 7         print(*matrix_set, sep=', ', end='; ')
 8
 9
10 def print_factor_set(factor_set_res):
11     print('Фактор-множество множества A по эквивалентности \u03B5: {',
       ↪ end='')
12     factor_set_res = [list(subset) for subset in factor_set_res]
13     for subset in factor_set_res[:-1:]:
14         print('{', end='')
15         print(*sorted(subset), sep=', ', end='}, ')
16     print('{', end='')
17     print(*sorted(factor_set_res[-1]), sep=', ', end='}}\n')
18
19
20 def factor_set(matrix, size):
21     classes = [[j + 1 for j, value in enumerate(matrix[i]) if value == 1] for
       ↪ i in range(size)]
22     return set(frozenset(subset) for subset in classes), classes
23
24
25 def full_system_of_class_representatives(factor, classes):
26     print('Полная система представителей классов эквивалентности \u03B5 на
       ↪ множестве A: T={', end='')
27     system = [min(subset) for subset in factor]
28     print(*system, sep=', ', end='} \u2282 A, где ')
29     eplison_numbers = []
30     for representative in system:
31         for i, class_ in enumerate(classes):
```

```
32              if representative in class_:
33                  eplison_numbers.append(i + 1)
34                  break
35      for i, number in enumerate(eplison_numbers[:-1:]):
36          print(f'{system[i]} \u2208 \u03B5({number}) = {classes[number -
            ↪ 1]}, ', end='')
37      print(f'{system[-1]} \u2208 \u03B5({eplison_numbers[-1]}) =
        ↪ {classes[eplison_numbers[-1] - 1]}')
38
39
40  def make_equivalent_closure(copy, size, matrix_set):
41
42      for u in range(size):
43          if copy[u][u] == 0:
44              copy[u][u] = 1
45          for k in range(size):
46              if copy[u][k] and not copy[k][u]:
47                  copy[k][u] = 1
48              for i in range(size):
49                  for j in range(size):
50                      if copy[k][i] == copy[i][j] == 1 and copy[k][j] == 0:
51                          copy[k][j] = 1
52
53      return [(i + 1, j + 1) for i in range(size) for j in range(size)
54              if copy[i][j] and (i + 1, j + 1) not in matrix_set], copy
55
56
57  def is_transitive(matrix, size):
58
59      for k in range(size):
60          for i in range(size):
61              for j in range(size):
62                  if matrix[k][i] == matrix[i][j] == 1 and matrix[k][j] == 0:
63                      return False
64      return True
65
66
67  def is_symmetric_or_antisymmetric(matrix, size):
68
69      flag_symmetric = True
70      flag_antisymmetric = True
```

```
71
72      for i in range(size):
73          for j in range(size):
74              if not matrix[i][j] == matrix[j][i]:
75                  flag_symmetric = False
76              if matrix[i][j] == matrix[j][i] == 1 and not i == j:
77                  flag_antisymmetric = False
78              if not flag_symmetric and not flag_antisymmetric:
79                  return False, False
80
81      return flag_symmetric, flag_antisymmetric
82
83
84  def is_reflexive_or_anti_reflexive(matrix, size):
85
86      flag_reflexive = True
87      flag_anti_reflexive = True
88
89      for i in range(size):
90          if matrix[i][i] == 0:
91              flag_reflexive = False
92          elif matrix[i][i] == 1:
93              flag_anti_reflexive = False
94          if not flag_reflexive and not flag_anti_reflexive:
95              return False, False
96
97      return flag_reflexive, flag_anti_reflexive
98
99
100 def get_data():
101     print('Введите размер матрицы:')
102     n = int(input())
103     print(f'Введите построчно элементы матрицы (по {n})')
104     m = [[int(elem) for elem in input().split()] for _ in range(n)]
105     return m, sorted([(i + 1, j + 1) for i in range(n) for j in range(n) if
        ↪ m[i][j] == 1]), n
106
107
108 def hasse_greater_eq(nums):
109     res = []
110     res.append((nums[0], 1, []))
```

```
111     for i, num in enumerate(nums[1:]):
112         res.append((num, res[-1][1] + 1, [res[-1][0]]))
113     return res
114 
115 
116 def hasse_division(dividers_num):
117     hasse_list = []
118     sl = {key: 1 for i, key in enumerate(dividers_num)}
119 
120     for number in dividers_num[1:]:
121         for divider in dividers_num[:dividers_num.index(number)]:
122             if number % divider == 0:
123                 sl[number] = sl[divider] + 1
124 
125     for k, v in sl.items():
126         pod_res = []
127         for k1, v1 in sl.items():
128             if v1 + 1 == v and k % k1 == 0:
129                 pod_res.append(k1)
130         hasse_list.append((k, v, pod_res))
131     hasse_list.sort(key=lambda x: x[1])
132     return hasse_list
133 
134 
135 def dividers(num, flag=False):
136     begin = 1
137     if flag:
138         begin = 2
139     return [divider for divider in range(begin, int(num / 2) + 1) if not num %
        ↪ divider] + [num]
140 
141 
142 def min_max_elements(lst):
143     if lst[0][1] == lst[1][1]:
144         print('Наименьшего элемента в данном множестве нет')
145     else:
146         print(f'Наименьший элемент множества: {lst[0][0]}')
147 
148     if lst[-1][1] == lst[-2][1]:
149         print('Наибольшего элемента в данном множестве нет')
150     else:
```

```
151         print(f'Наибольший элемент множества: {lst[-1][0]}')
152
153     print(f'Минимальные элементы множества: {lst[0][0]}, ', end='')
154     minimum = lst[0][1]
155     for values in lst[1:]:
156         if values[1] == minimum:
157             print(values[0], end=', ')
158         else:
159             break
160     print('\n')
161     print(f'Максимальные элементы множества: {lst[-1][0]}, ', end='')
162     maximum = lst[-1][1]
163     for values in lst[-2::-1]:
164         if values[1] == maximum:
165             print(values[0], end=', ')
166         else:
167             break
168     print('\n')
169
170
171 print('Вы хотите получить фактор-множество отношения и полную систему
    ↪ представителей классов? Да (1) или Нет (0)')
172 yes_or_no = int(input())
173 if yes_or_no:
174     matrix, matrix_set, size = get_data()
175     print_matrix_set(matrix_set)
176     print('\n')
177     print('Свойства бинарного отношения:')
178     flagT = True
179     flagR = True
180     flagS = True
181
182     if is_transitive(matrix, size):
183         print('Отношение является транзитивным')
184     else:
185         print('Отношение не является транзитивным')
186         flagT = False
187
188     symm, _ = is_symmetric_or_antisymmetric(matrix,size)
189     if symm:
190         print('Отношение является симметричным')
```

```
191     else:
192         print('Отношение не является симметричным')
193         flagS = False
194 
195     refl, _ = is_reflexive_or_anti_reflexive(matrix, size)
196     if refl:
197         print('Отношение является рефлексивным')
198     else:
199         print('Отношение не является рефлексивным')
200         flagR = False
201 
202     print('\n')
203     if not flagS or not flagR or not flagT:
204         print('Так как отношение не обладает свойством ', end='')
205         if not flagS:
206             print('симметричности', end=', ')
207         if not flagT:
208             print('транзитивности', end=', ')
209         if not flagR:
210             print('рефлексивности', end=', ')
211         print('то для получения фактор-множества отношения, требуется
            ↪  построить эквивалентное замыкание.')
212 
213         copy = matrix
214         ls, mt = make_equivalent_closure(copy, size, matrix_set)
215 
216         print('Эквивалентное замыкание бинарного отношения: {', end='')
217         print(*ls, sep=', ', end='}\n\n')
218 
219         print('Матрица эквивалентного замыкания бинарного отношения:')
220         for i in range(len(mt)):
221             print(*mt[i])
222         print('\n')
223 
224         factor_set_res, classes = factor_set(matrix, size)
225         print_factor_set(factor_set_res)
226         full_system_of_class_representatives(factor_set_res, classes)
227 
228     else:
229         print('Заданное отношение является эквивалентным. Его матрица:')
230         for i in range(len(matrix)):
```

```
231             print(*matrix[i])
232         print('\n')
233 
234         factor_set_res, classes = factor_set(matrix, size)
235         print_factor_set(factor_set_res)
236         full_system_of_class_representatives(factor_set_res, classes)
237 
238 
239 print('Вы хотите получить минимальные/наименьшие и максимальные/наибольшие
    ↪ элементы множества? Да (1) или Нет (0)')
240 yes_or_no = int(input())
241 res = None
242 if yes_or_no:
243     print('Выберите тип задания множества: число (1) или заданное множество
        ↪ (2)')
244     set_type = int(input())
245 
246     print('Выбереге тип порядка: <= (1) или отношение делимости (2)')
247     order_type = int(input())
248 
249     if set_type == 1:
250         print('Введите число')
251         num = int(input())
252         print('Хотите ли добавить единицу во множество? Да(1), Нет(0)')
253         yes_or_no = int(input())
254         sub_res = None
255         if yes_or_no == 1:
256             if order_type == 2:
257                 sub_res = dividers(num)
258                 res = hasse_division(sub_res)
259             else:
260                 sub_res = [i + 1 for i in range(num)]
261                 res = hasse_greater_eq(sub_res)
262         else:
263             if order_type == 2:
264                 sub_res = dividers(num, True)
265                 res = hasse_division(sub_res)
266             else:
267                 sub_res = [i + 2 for i in range(num - 1)]
268                 res = hasse_greater_eq(sub_res)
269     else:
```

```
270         print('Введите множество')
271         num = [int(value) for value in input().split()]
272         num = list(set(num))
273         num.sort()
274 
275         if order_type == 2:
276             res = hasse_division(num)
277         elif order_type == 1:
278             res = hasse_greater_eq(num)
279 
280     min_max_elements(res)
281     print('Вы хотите получить диаграмму Хассе? Да(1) или Нет(0)')
282     yes_or_no = int(input())
283     if yes_or_no:
284         import hasse_visualization as hv
285         if order_type == 1:
286             hv.visual(res, True)
287         else:
288             hv.visual(res)
289     print(res)
290 
291 print('Вы хотите получить элементы решетки концептов C(K)? Да (1) или Нет
    ↪ (0)')
292 yes_or_no = int(input())
293 if yes_or_no:
294     import lattice_of_concepts as lc
295     lc.main()
296 '''
297 Примеры входных данных для 1-ой части работы:
298 
299 3
300 0 1 0
301 0 0 1
302 1 0 0
303 
304 4
305 1 0 1 0
306 1 1 0 0
307 0 0 1 0
308 0 1 0 1
309 
```

```
5
1 0 1 1 0
0 1 0 1 0
1 0 1 1 0
1 1 1 1 0
0 0 0 0 1

8
0 1 1 0 0 0 0 0
1 0 1 0 0 0 0 0
0 1 1 0 0 0 0 0
0 0 0 1 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 1 1
0 0 0 0 0 1 1 0
0 0 0 0 0 1 1 1
```

Примеры входных данных для 2-ой части работы:

```
1
2
30
1
1

2
2
2 3 21 15 14 4 8 30 16 32
1
```

Пример входных данных для 3-ей части работы:

```
1 2 3 4

a b c d

1 0 1 0
1 1 0 0
0 1 0 1
0 1 0 1
'''
```

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были рассмотрены понятия эквивалентного замыкания бинарного отношения и получения представителей фактор-множества. Также были получены алгоритмы вычисления минимальных и максимальных, и наименьших и наибольших элементов бинарного отношения, а также был определен и программно реализован алгоритм построения диаграммы Хассе. Был описан алгоритм построения решетки концептов. Для всех алгоритмов произведена асимптотическая оценка.