

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.

\_\_\_\_\_  
подпись, дата

В. А. Молчанов

Саратов 2022

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
1 <b>Цель работы и порядок ее выполнения</b> .....	4
2 Теоретические сведения .....	5
2.1 Эквивалентное замыкание бинарного отношения .....	5
2.1.1 Определение эквивалентного замыкания отношения .....	5
2.1.2 Алгоритм построения эквивалентного замыкания бинарного отношения .....	5
2.2 Фактор-множество отношения .....	6
2.2.1 Определение среза отношения через элемент .....	6
2.2.2 Определение фактор-множества отношения .....	6
2.2.3 Алгоритм построения фактор-множества бинарного отношения .....	6
3 Программная реализация рассмотренных алгоритмов .....	7
3.1 Результаты тестирования программы .....	7
3.2 Коды программ, реализующих рассмотренные алгоритмы .....	10
3.2.1 Код программы, реализующей визуализацию диаграммы Хассе .....	10
3.2.2 Код программы, реализующей получение решетки концептов .....	14
3.2.3 Код программы, реализующей основные алгоритмы .....	16
ЗАКЛЮЧЕНИЕ .....	25

## **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 Определение эквивалентного замыкания отношения

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

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

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

Итак, исходя из вышесказанного, можно сделать вывод, что 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.  $f_r(\rho) = \rho \cup \Delta_A$  – наименьшее рефлексивное бинарное отношение, содержащее отношение  $\rho \subset A^2$ .
2.  $f_s(\rho) = \rho \cup \rho^{-1}$  – наименьшее симметричное бинарное отношение, содержащее отношение  $\rho \subset A^2$ .
3.  $f_t(\rho) = \bigcup_{n=1}^{\infty} \rho^n$  – наименьшее транзитивное бинарное отношение, содержащее отношение  $\rho \subset A^2$ .
4.  $f_{eq}(\rho) = f_t f_s f_r(\rho)$  – наименьшее отношение эквивалентности, содержащее отношение  $\rho \subset A^2$ .

#### 2.1.2 Алгоритм построения эквивалентного замыкания бинарного отношения

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

*Выход.* Эквивалентное замыкание бинарного отношения.

1. Создать пустой список для хранения пар замыкания.
  - а) Цикл по  $i$  от 1 до  $N$ .
    1. Если  $M_{ii} = 0$ , пару  $(i, i)$  добавить в замыкание.
  - б) Цикл по  $i$  от 1 до  $N$ , цикл по  $j$  от 1 до  $N$ .
    1. Если  $M_{ij} = 1$  и  $M_{ji} = 0$ , добавить пару  $(j, i)$  в замыкание.

с) Цикл по  $e$  от 1 до  $N$ , цикл по  $k$  от 1 до  $N$ , цикл по  $i$  от 1 до  $N$ , цикл по  $j$  от 1 до  $N$ .

1. Если  $M_{ki} = M_{i,j} = 1$  и  $M_{kj} = 0$ , то добавить пару  $(k, k)$  в замыкание транзитивности и замыкание эквивалентности.

2. Ответ — эквивалентное замыкание бинарного отношения  $\rho$ .

Трудоемкость алгоритма  $O(N + N^2 + N^4) = O(N^4)$

## 2.2 Фактор-множество отношения

### 2.2.1 Определение среза отношения через элемент

Для любого подмножества  $X \subset A$  множество:

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

называется *образом* множества  $X$  относительно отношения  $\rho$ .

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

### 2.2.2 Определение фактор-множества отношения

Эквивалентное бинарное отношение на множестве  $A$  также принято обозначать как  $\varepsilon$ .

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

Множество всех таких классов эквивалентности  $\{[a] : a \in A\}$  называется *фактор-множеством* множества  $A$  по эквивалентности  $\varepsilon$  и обозначается  $A/\varepsilon$ .

### 2.2.3 Алгоритм построения фактор-множества бинарного отношения

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

*Выход.* Фактор-множество отношения.

1. Создать  $N$  пустых списков.

а) Цикл по  $i$  от 1 до  $N$ , цикл по  $j$  от 1 до  $N$ .

1. Если  $M_{ij} = 1$  добавить  $j$  в список с номером  $i$ .

2. Ответ — фактор-множество отношения.

Трудоемкость алгоритма  $O(N^2)$

## 3 Программная реализация рассмотренных алгоритмов

### 3.1 Результаты тестирования программы

```
Исходное отношение: {(1, 1), (1, 3), (1, 4), (2, 2), (2, 4), (3, 1), (3, 3), (3, 4), (4, 1), (4, 2), (4, 3), (4, 4), (5, 5)}

Свойства бинарного отношения:
Отношение не является транзитивным
Отношение является симметричным
Отношение является рефлексивным

Так как отношение не обладает свойством транзитивности, то для получения фактор-множества отношения, требуется построить эквивалентное замыкание.
Эквивалентное замыкание бинарного отношения: {(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4), (3, 1), (3, 2), (3, 3), (3, 4), (4, 1), (4, 2), (4, 3), (4, 4), (5, 5)}

Матрица эквивалентного замыкания бинарного отношения:
1 1 1 1 0
1 1 1 1 0
1 1 1 1 0
1 1 1 1 0
0 0 0 0 1

Фактор-множество множества A по эквивалентности  $\epsilon$ : {{5}, {1, 2, 3, 4}}
Полная система представителей классов эквивалентности  $\epsilon$  на множестве A: T={5, 1}=A
```

Рисунок 1

```
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
Выберите тип порядка: <= (1) или отношение делимости (2)
1
Введите число
30
Хотите ли добавить единицу во множество? Да(1), Нет(0)
1
Наименьший элемент множества: 1
Наибольший элемент множества: 30
Минимальные элементы множества: 1,
Максимальные элементы множества: 30,

Вы хотите получить диаграмму Хассе? Да(1) или Нет(0)
1
[(1, 1, []), (2, 2, [1]), (3, 2, [1]), (5, 2, [1]), (6, 3, [2, 3]), (10, 3, [2, 5]), (15, 3, [3, 5]), (30, 4, [6, 10, 15])]
```

Рисунок 2

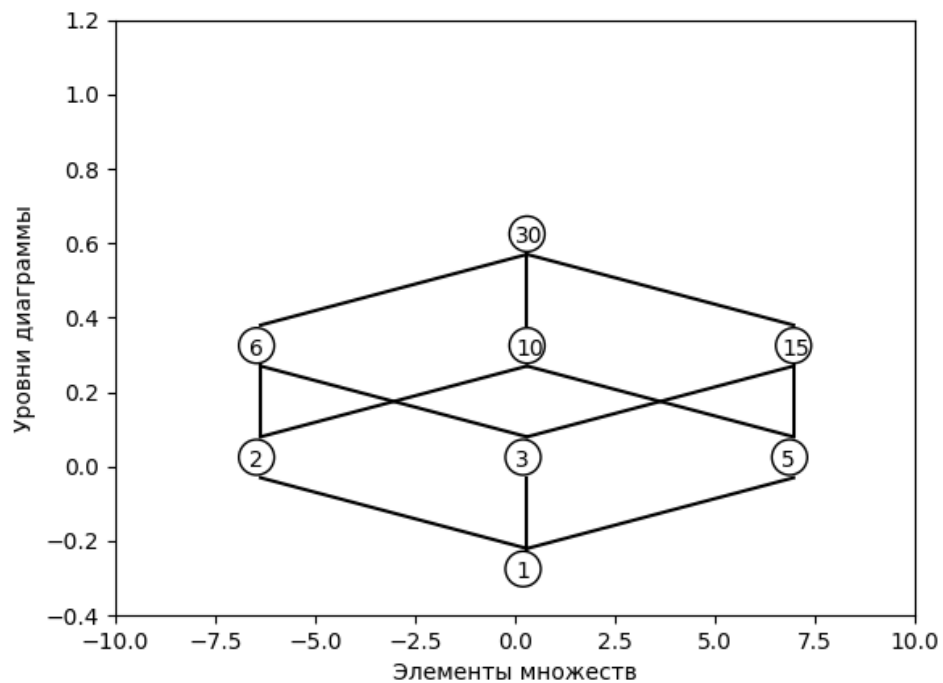


Рисунок 3

```

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
Выберите 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



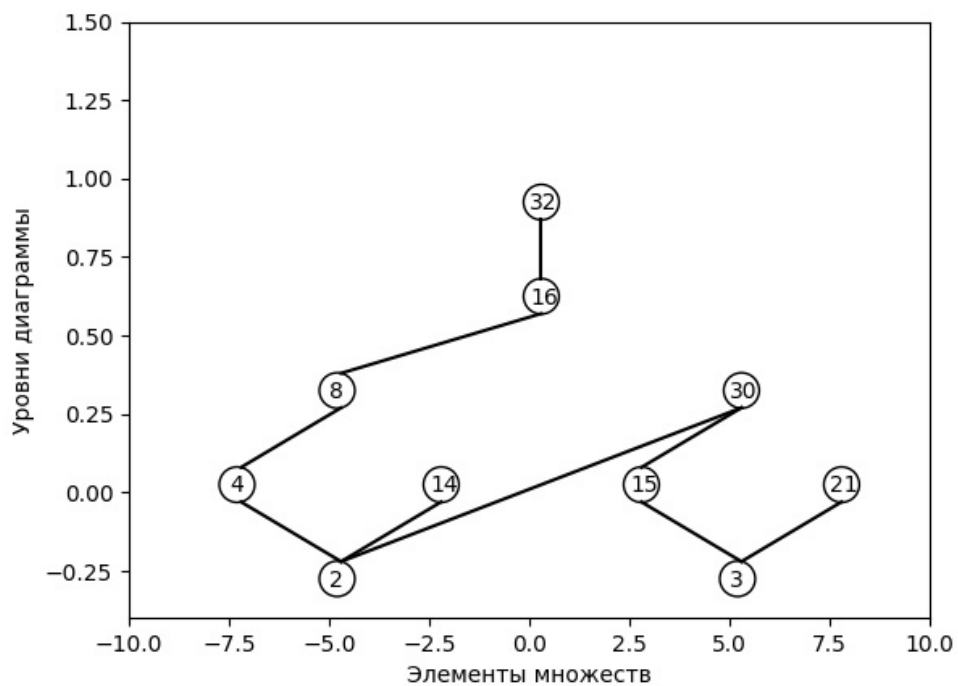


Рисунок 5

```

Вы хотите получить минимальные/наименьшие и максимальные/наибольшие элементы множества? Да (1) или Нет (0)
Введите тип задания множества: число (1) или заданное множество (2)
Выберите тип порядка: <= (1) или отношение делимости (2)
Введите число
Хотите ли добавить единицу во множество? Да(1), Нет(0)

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

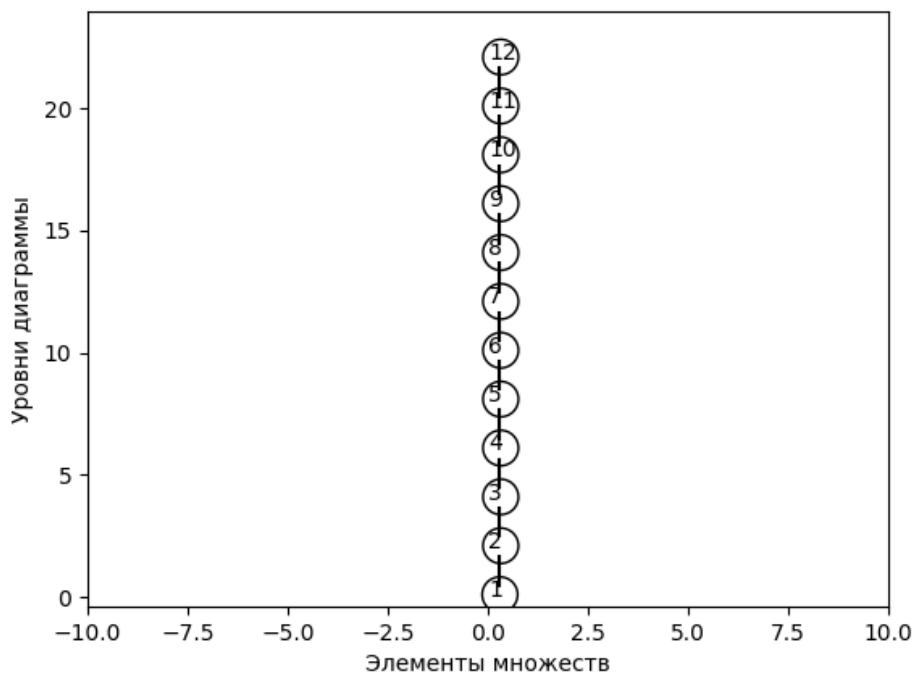


Рисунок 7

```

Вы хотите получить элементы решетки концептов C(K)? Да (1) или Нет (0)
1
Введите множество объектов
1 2 3 4
Введите множество атрибутов
a b c d
Введите значения матрицы бинарного отношения построчно (по 4)
1 0 1 0
1 1 0 0
0 1 0 1
0 1 0 1
1 2 3 4
1 1 0 1 0
2 1 1 0 0
3 0 1 0 1
4 0 1 0 1
Решетка концептов C(K) состоит из элементов: ({1,2},{a}), ({2},{a, b}), ({2,3,4},{b}), ({1},{a, c}), ({3,4},{b, d}), (G, ∅), (∅, M)

```

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=False):
32
33     plt.xlim(-10.0, 10.0)
34     lim = lst[-1][1]
35     plt.xlabel('Элементы множеств')
36     plt.ylabel('Уровни диаграммы')
37
38     if not flag:
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,
67                 ↪ facecolors='none', edgecolors='black')
68             x_value += delta
69             y_value += dy
70             current_level += 1
71
72     y_value = y_save + dy
73     for level, values in enumerate(levels_numbers_list[1:]):
74         level += 1
75         x_value = x_save
76         delta = 20 / len_levels[level]
77         delta1 = delta / 2
78         x_value += delta1
79         current_level = level
80         for value in values:
81             dividers_lst = dividers(value)
82             y_previous = dy
83             for i, values_levels in enumerate(levels_numbers_list[level -
84                 ↪ 1::-1]):
85                 x_value1 = x_save
86                 delta_value1 = 20 / len_levels[current_level - i - 1]
87                 delta1_value1 = delta_value1 / 2
88                 x_value1 += delta1_value1
89                 for value1 in values_levels:
90                     if value % value1 == 0 and value1 in dividers_lst:
91                         value1_dividers = dividers(value1)

```

```

90         dividers_lst = [val for val in dividers_lst if val
91             ↪ not in value1_dividers + [value1]]
92         plt.plot([x_value1 + 0.3, x_value + 0.3], [y_value
93             ↪ - y_previous + 0.08, y_value - 0.03],
94                 color='black')
95         x_value1 += delta_value1
96         y_previous += dy
97         x_value += delta
98         y_value += dy
99     else:
100         plt.ylim(-0.4, 2 * lim)
101         x_value = 0
102         y_value = 0
103         dy = 2 * lim / len(lst)
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',
106             ↪ edgecolors='black')
107         y_value += dy
108         for value in lst[1:]:
109             plt.text(x_value, y_value, f '{value[0]} ')
110             plt.scatter(x_value + 0.32, y_value + 0.1, s=250,
111                 ↪ facecolors='none', edgecolors='black')
112             plt.plot([x_value + 0.3, x_value + 0.3], [y_value - 0.35, y_value
113                 ↪ - dy + 0.5], color='black')
114             y_value += dy
115         plt.show()
116
117     '''
118     Примеры входных данных:
119     visual([(1, 1, []), (2, 2, [1]), (3, 3, [2]), (4, 4, [3]), (5, 5, [4]), (6, 6,
120         ↪ [5]), (7, 7, [6]), (8, 8, [7]),
121         (9, 9, [8]), (10, 10, [9]), (11, 11, [10]), (12, 12, [11])], True)
122     visual([(1, 1, []), (2, 2, [1]), (3, 3, [2]), (4, 4, [3]), (5, 5, [4]), (6, 6,
123         ↪ [5]), (7, 7, [6])], True)

```

```

124 visual([(1, 1, []), (2, 2, [1]), (3, 2, [1]), (5, 2, [1]), (6, 3, [2, 3]),
    ↪ (10, 3, [2, 5]), (15, 3, [3, 5]),
125 (30, 4, [6, 10, 15]))])
126 '''

```

### 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
55
56 def print_lattice_of_concepts(mat, attr):
57     print('Решетка концептов C(K) состоит из элементов: ', end='')
58     lattice_of_concepts, g = get_lattice_of_concepts(mat, len(attr),
        → list(attr.keys()))
59     for key, value in lattice_of_concepts.items():
60         value = list(value)
61         print('{', end='')
62         print(*value, sep=', ', end='}, ')
63         print('{ ' + key + '}', end='), ')
64
65     print(g, end=', ')
66     print('(\u2205, M)')
67
68
69 def main():
70     print('Введите множество объектов')
71     obj = [int(value) for value in input().split()]
72
73     print('Введите множество атрибутов')
74     attr = input().split()

```

```

75
76     obj, attr = make_nums_obj_attr(obj, attr)
77
78     mat = get_matrix(len(attr))
79
80     print_matrix(mat, obj)
81     print_lattice_of_concepts(mat, attr)

```

### 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: {',
12           '\u2192 end='')
13     factor_set_res = [list(subset) for subset in factor_set_res]
14     for subset in factor_set_res[:-1]:
15         print('{', end='')
16         print(*sorted(subset), sep=', ', end='}, ')
17     print('{', end='')
18     print(*sorted(factor_set_res[-1]), sep=', ', end='}} \n')
19
20 def factor_set(matrix, size):
21     classes = [[j + 1 for j, value in enumerate(matrix[i]) if value == 1] for
22                '\u2192 i in range(size)]
23     return set(frozenset(subset) for subset in classes), classes
24
25 def full_system_of_class_representatives(factor, classes):
26     print('Полная система представителей классов эквивалентности \u03B5 на
27           '\u2192 множестве A: T={', end='')
28     system = [min(subset) for subset in factor]
29     print(*system, sep=', ', end='} \u2282 A, где ')
30     eplison_numbers = []
31     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
106         ↪ m[i][j] == 1]), n
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 %
140             ↪ divider] + [num]
141
142
143 def min_max_elements(lst):
144     if lst[0][1] == lst[1][1]:
145         print('Наименьшего элемента в данном множестве нет')
146     else:
147         print(f'Наименьший элемент множества: {lst[0][0]}')
148
149     if lst[-1][1] == lst[-2][1]:
150         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('Выберете 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
310 5
311 1 0 1 1 0
312 0 1 0 1 0
313 1 0 1 1 0
314 1 1 1 1 0
315 0 0 0 0 1
316
317 8
318 0 1 1 0 0 0 0 0
319 1 0 1 0 0 0 0 0
320 0 1 1 0 0 0 0 0
321 0 0 0 1 1 0 0 0
322 0 0 0 0 1 0 0 0
323 0 0 0 0 0 1 1 1
324 0 0 0 0 0 1 1 0
325 0 0 0 0 0 1 1 1
326
327 Примеры входных данных для 2-ой части работы:
328
329 1
330 2
331 30
332 1
333 1
334
335 2
336 2
337 2 3 21 15 14 4 8 30 16 32
338 1
339
340 Пример входных данных для 3-ей части работы:
341 1 2 3 4
342
343 a b c d
344
345 1 0 1 0
346 1 1 0 0
347 0 1 0 1
348 0 1 0 1
349 ' ' '

```



## **ЗАКЛЮЧЕНИЕ**

В ходе лабораторной работы были рассмотрены понятия эквивалентного замыкания бинарного отношения и получения представителей фактормножества. 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.