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 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. $f_r(\rho) = \rho \cup \triangle_A$ – 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 $\rho \subset A^2$.

2.1.2 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.
   a) Цикл по i от 1 до N.
      1. Если $M_{ii} = 0$, пару (i, i) добавить в замыкание.
   b) 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.

c) 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 $M_{ki} = 0$, то добавить пару (k, k) в замыкание транзитивности и замыкание эквивалентности.

2. Ответ — эквивалентное замыкание бинарного отношения ρ.

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:

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

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 N пустых списков.

   a) 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

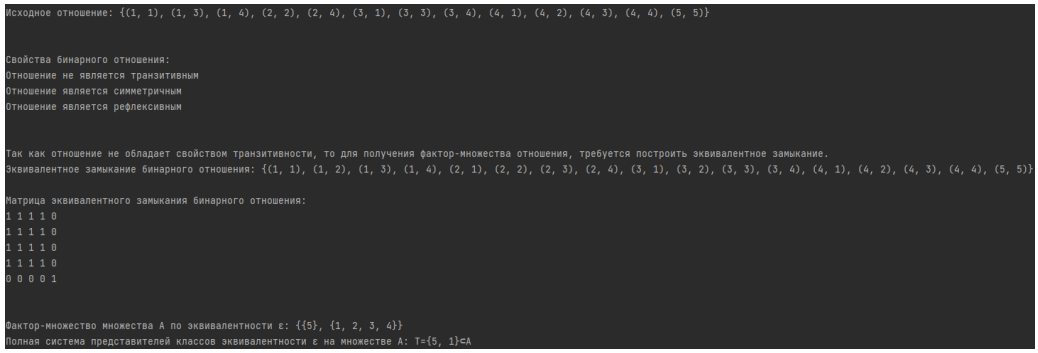

Рисунок 1

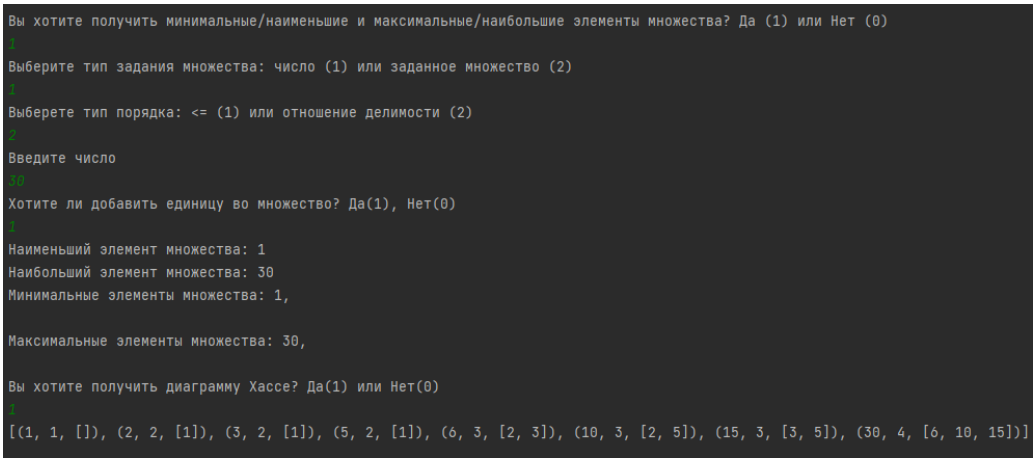

Рисунок 2

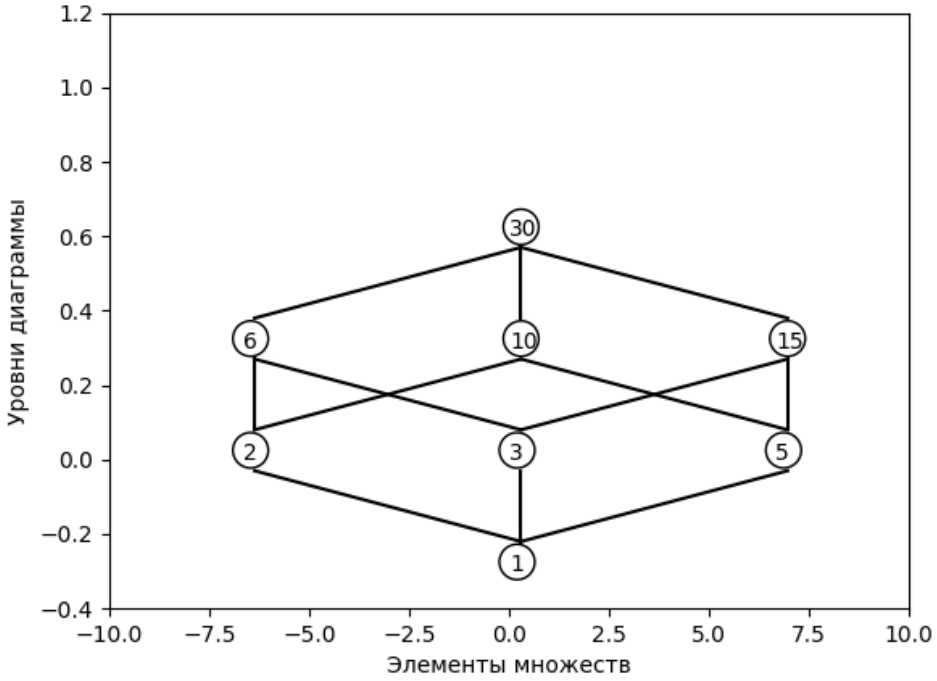

Рисунок 3

```
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 4 3 6 8 16 30 15
Наименьшего элемента в данном множестве нет
Наибольший элемент множества: 16
Минимальные элементы множества: 2, 3,

Максимальные элементы множества: 16,

Вы хотите получить диаграмму Хассе? Да(1) или Нет(0)
1
[(2, 1, []), (3, 1, []), (4, 2, [2]), (6, 2, [2, 3]), (15, 2, [3]), (8, 3, [4]), (30, 3, [6, 15]), (16, 4, [8])]
```

Рисунок 4

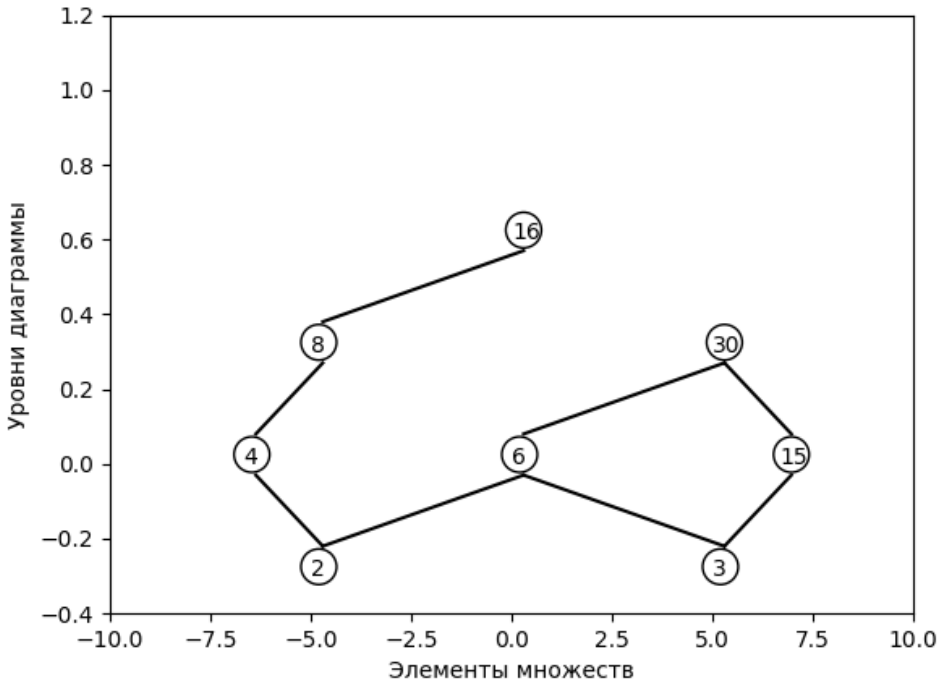

Рисунок 5

```
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

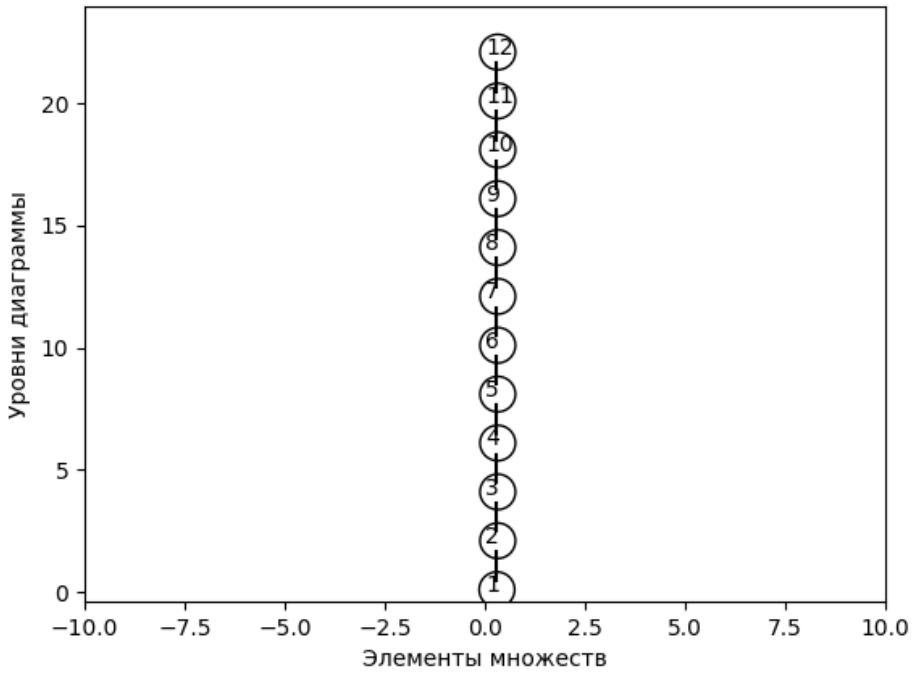

Рисунок 7

3.2 Коды программ, реализующих рассмотренные алгоритмы

3.2.1 Код программы, реализующей визуализацию диаграммы Хассе

```
import matplotlib.pyplot as plt


def levels_length(lst):
    max_len = 1
    max_level_length_list = []
    value = lst[0][1]
    for values in lst[1:]:
        if values[1] == value:
            max_len += 1
        if values[1] != value:
            max_level_length_list.append(max_len)
            max_len = 1
            value = values[1]

    max_level_length_list.append(max_len)
    return max_level_length_list


def visual(lst, flag=False):
```

```
21      plt.xlim(-10.0, 10.0)
22      lim = lst[-1][1]
23      plt.xlabel('Элементы множеств')
24      plt.ylabel('Уровни диаграммы')
25
26      if not flag:
27          len_levels = levels_length(lst)
28
29          levels_numbers_list = [[] for _ in range(len(len_levels))]
30          for value in lst:
31              levels_numbers_list[value[1] - 1].append(value[0])
32
33          plt.ylim(-0.4, 1.2 * lim / 4)
34
35          x = -10
36          x_save = -10
37
38          y = -0.3
39          y_save = y
40
41          current_level = 0
42
43          delta = 20 / len_levels[current_level]
44          delta1 = delta / 2
45          x += delta1
46
47          dy = 0.3
48          dx = 0.2
49          plt.text(x, y, f'{lst[0][0]}')
50          plt.scatter(x + 0.2, y + 0.025, s=250, facecolors='none',
            ↪ edgecolors='black')
51          x += delta
52          for i, value in enumerate(lst[1:]):
53              if lst[i+1][1] == lst[i][1]:
54
55                  plt.text(x, y, f'{value[0]}')
56
57                  if value[0] > 9:
58                      dx = 0.3
59                  if value[0] > 99:
60                      dx = 0.5
```

```
61                 plt.scatter(x + dx, y + 0.025, s=250, facecolors='none',
                   ↪ edgecolors='black')
62
63                 dx = 0.2
64                 x += delta
65
66             if lst[i][1] != lst[i+1][1]:
67                 current_level += 1
68                 x = x_save
69                 delta = 20 / len_levels[current_level]
70                 delta1 = delta / 2
71                 x += delta1
72                 y += dy
73                 if value[0] > 9:
74                     dx = 0.3
75                 if value[0] > 99:
76                     dx = 0.5
77
78                 plt.text(x, y, f'{value[0]}')
79                 plt.scatter(x + dx, y + 0.025, s=250, facecolors='none',
                   ↪ edgecolors='black')
80                 dx = 0.2
81                 x += delta
82
83         y = y_save + dy
84         for level, values in enumerate(levels_numbers_list[1:]):
85             level += 1
86             x = x_save
87             delta = 20 / len_levels[level]
88             delta1 = delta / 2
89             x += delta1
90             for value in values:
91                 x_value1 = x_save
92                 delta_value1 = 20 / len_levels[level - 1]
93                 delta1_value1 = delta_value1 / 2
94                 x_value1 += delta1_value1
95                 for value1 in levels_numbers_list[level - 1]:
96                     if value % value1 == 0:
97                         plt.plot([x_value1 + 0.3, x + 0.3], [y - dy + 0.08, y
                           ↪ - 0.03], color='black')
98                     x_value1 += delta_value1
```

```
 99                 x += delta
100             y += dy
101     else:
102         plt.ylim(-0.4, 2 * lim)
103 
104         x = 0
105         y = 0
106         dy = 2 * lim / len(lst)
107 
108         plt.text(x, y, f'{lst[0][0]}')
109         plt.scatter(x + 0.3, y + 0.1, s=250, facecolors='none',
            ↪ edgecolors='black')
110         y += dy
111         for value in lst[1:]:
112             plt.text(x, y, f'{value[0]}')
113             plt.scatter(x + 0.32, y + 0.1, s=250, facecolors='none',
                ↪ edgecolors='black')
114             plt.plot([x + 0.3, x + 0.3], [y - 0.35, y - dy + 0.5],
                ↪ color='black')
115             y += dy
116 
117     plt.show()
118 
119 '''
120 visual([(1, 1, []), (2, 2, [1]), (3, 3, [2]), (4, 4, [3]), (5, 5, [4]), (6, 6,
    ↪ [5]), (7, 7, [6]), (8, 8, [7]),
121         (9, 9, [8]), (10, 10, [9]), (11, 11, [10]), (12, 12, [11])], True)
122 
123 visual([(1, 1, []), (2, 2, [1]), (3, 3, [2]), (4, 4, [3]), (5, 5, [4]), (6, 6,
    ↪ [5]), (7, 7, [6])], True)
124 
125 visual([(1, 1, []), (2, 2, [1]), (3, 2, [1]), (5, 2, [1]), (6, 3, [2, 3]),
    ↪ (10, 3, [2, 5]), (15, 3, [3, 5]),
126          (30, 4, [6, 10, 15])])
127 '''
```

3.2.2 Код программы, реализующей получение решетки концептов

```
1 def make_nums_obj_attr(objects, attributes):
2     return {value: i for i, value in enumerate(objects)}, {value: i for i,
      ↪ value in enumerate(attributes)}
3 
4 
```

```
 5  def get_rho_minus_attr(matrix, size):
 6      closure_system = set()
 7      for i in range(size):
 8          new_subset = []
 9          for j in range(size):
10              if matrix[j][i] == 1:
11                  new_subset.append(j + 1)
12          new_subset = frozenset(new_subset)
13          if not closure_system:
14              closure_system.add(new_subset)
15          else:
16              sets = []
17              for subset in closure_system:
18                  new_subset = frozenset(subset.intersection(new_subset))
19                  sets.append(new_subset)
20              for subset in sets:
21                  closure_system.add(subset)
22              closure_system.add(new_subset)
23      return closure_system
24
25
26  def get_matrix(size):
27      print(f'Введите значения матрицы бинарного отношения построчно (по
        ↪ {size})')
28      return [[int(value) for value in input().split()] for _ in range(size)]
29
30
31  def print_matrix(mat, obj):
32      print('  ', end='')
33      print(*obj)
34      symbols = list(obj.keys())
35      for i in range(len(mat)):
36          print(symbols[i], end=' ')
37          print(*mat[i])
38
39  # print('Введите множество объектов')
40  # obj = [int(value) for value in input().split()]
41  #
42  # print('Введите множество атрибутов')
43  # attr = input().split()
44  #
```

```
# obj, attr = make_nums_obj_attr(obj, attr)
#
# mat = get_matrix(len(attr))
#
# print_matrix(mat, obj)
# print(get_rho_minus_attr(mat, len(attr)))

'''
1 2 3 4

a b c d

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

3.2.3 Код программы, реализующей основные алгоритмы

```
def matrix_set_view(matrix_set, flag=None):
    if not flag:
        print('Исходное отношение: {', end='')
        print(*matrix_set, sep=', ', end='}\n')
    else:
        print('{', end='')
        print(*matrix_set, sep=', ', end='; ')


def print_factor(factor_set_res):
    print('Фактор-множество множества A по эквивалентности \u03B5: {',
    ↪ end='')
    factor_set_res = [list(subset) for subset in factor_set_res]
    for subset in factor_set_res[:-1:]:
        print('{', end='')
        print(*subset, sep=', ', end='}, ')
    print('{', end='')
    print(*factor_set_res[-1], sep=', ', end='}}\n')


def factor_set(matrix, size):
    classes = [[j + 1 for j, value in enumerate(matrix[i]) if value == 1]
                                                    for i in range(size)]
```

```
23      return set(frozenset(subset) for subset in classes)
24
25
26  def full_system_of_class_representatives(factor):
27      res = []
28      for subset in factor:
29          res.append(min(subset))
30      print('''Полная система представителей классов эквивалентности
31                u03B5 на множестве A: T={', end='''''')
32      print(*res, sep=', ', end='} \u2282 A \n ')
33
34
35  def make_equivalent_closure(copy, size):
36      list_for_equivalent_closure = set()
37      for i in range(size):
38          for j in range(size):
39              if matrix[i][j] == 1 and matrix[j][i] == 0:
40                  copy[j][i] = 1
41              if copy[j][i]:
42                  list_for_equivalent_closure.add((j + 1, i + 1))
43          if matrix[i][i] == 0:
44              copy[i][i] = 1
45          if copy[i][i]:
46              list_for_equivalent_closure.add((i + 1, i + 1))
47
48      for _ in range(size):
49          for k in range(size):
50              for i in range(size):
51                  for j in range(size):
52                      if copy[k][i] == copy[i][j] == 1 and copy[k][j] == 0:
53                          copy[k][j] = 1
54                      if copy[k][j]:
55                          list_for_equivalent_closure.add((k + 1, j + 1))
56
57      return sorted(list_for_equivalent_closure), copy
58
59
60  def is_transitive(matrix, size):
61
62      for k in range(size):
63          for i in range(size):
```

```
64              for j in range(size):
65                  if matrix[k][i] == matrix[i][j] == 1 and matrix[k][j] == 0:
66                      return False
67      return True
68
69
70  def is_symmetric_or_antisymmetric(matrix, size):
71
72      flag_symmetric = True
73      flag_antisymmetric = True
74
75      for i in range(size):
76          for j in range(size):
77              if not matrix[i][j] == matrix[j][i]:
78                  flag_symmetric = False
79              if matrix[i][j] == matrix[j][i] == 1 and not i == j:
80                  flag_antisymmetric = False
81              if not flag_symmetric and not flag_antisymmetric:
82                  return False, False
83
84      return flag_symmetric, flag_antisymmetric
85
86
87  def is_reflexive_or_anti_reflexive(matrix, size):
88
89      flag_reflexive = True
90      flag_anti_reflexive = True
91
92      for i in range(size):
93          if matrix[i][i] == 0:
94              flag_reflexive = False
95          elif matrix[i][i] == 1:
96              flag_anti_reflexive = False
97          if not flag_reflexive and not flag_anti_reflexive:
98              return False, False
99
100     return flag_reflexive, flag_anti_reflexive
101
102
103 def get_data():
104     print('Введите размер матрицы:')
```

```
105     n = int(input())
106     print(f'Введите построчно элементы матрицы (по {n})')
107     m = [[int(elem) for elem in input().split()] for _ in range(n)]
108     m_set = [(i + 1, j + 1) for i in range(n) for j in range(n) if m[i][j] ==
        ↪ 1]
109     return m, sorted(m_set), n
110
111
112 def hasse_greater_eq(nums):
113     res = []
114     res.append((nums[0], 1, []))
115     for i, num in enumerate(nums[1:]):
116         res.append((num, res[-1][1] + 1, [res[-1][0]]))
117     return res
118
119
120 def hasse_division(dividers_num):
121     hasse_list = []
122     sl = {key: 1 for i, key in enumerate(dividers_num)}
123
124     for number in dividers_num[1:]:
125         for divider in dividers_num[:dividers_num.index(number)]:
126             if number % divider == 0:
127                 sl[number] = sl[divider] + 1
128
129     for k, v in sl.items():
130         pod_res = []
131         for k1, v1 in sl.items():
132             if v1 + 1 == v and k % k1 == 0:
133                 pod_res.append(k1)
134         hasse_list.append((k, v, pod_res))
135     hasse_list.sort(key=lambda x: x[1])
136     return hasse_list
137
138
139 def dividers(num, flag=False):
140     result = []
141     begin = 1
142     if flag:
143         begin = 2
144     for i in range(begin, int(num / 2) + 1):
```

```
145         if num % i == 0:
146             result.append(i)
147     result.append(num)
148     return result
149
150
151 def min_max_elements(lst):
152     if lst[0][1] == lst[1][1]:
153         print('Наименьшего элемента в данном множестве нет')
154     else:
155         print(f'Наименьший элемент множества: {lst[0][0]}')
156
157     if lst[-1][1] == lst[-2][1]:
158         print('Наибольшего элемента в данном множестве нет')
159     else:
160         print(f'Наибольший элемент множества: {lst[-1][0]}')
161
162     print(f'Минимальные элементы множества: {lst[0][0]}, ', end='')
163     minimum = lst[0][1]
164     for values in lst[1:]:
165         if values[1] == minimum:
166             print(values[0], end=', ')
167         else:
168             break
169     print('\n')
170     print(f'Максимальные элементы множества: {lst[-1][0]}, ', end='')
171     maximum = lst[-1][1]
172     for values in lst[-2::-1]:
173         if values[1] == maximum:
174             print(values[0], end=', ')
175         else:
176             break
177     print('\n')
178     return
179
180
181 print('''Вы хотите получить фактор-множество отношения и полную систему
182         представителей классов? Да (1) или Нет (0)''')
183
184 yes_or_no = int(input())
185 if yes_or_no:
```

```
186     matrix, matrix_set, size = get_data()
187     matrix_set_view(matrix_set)
188     print('\n')
189     print('Свойства бинарного отношения:')
190     flagT = True
191     flagR = True
192     flagS = True
193
194     if is_transitive(matrix, size):
195         print('Отношение является транзитивным')
196     else:
197         print('Отношение не является транзитивным')
198         flagT = False
199
200     symm, _ = is_symmetric_or_antisymmetric(matrix,size)
201     if symm:
202         print('Отношение является симметричным')
203     else:
204         print('Отношение не является симметричным')
205         flagS = False
206
207     refl, _ = is_reflexive_or_anti_reflexive(matrix, size)
208     if refl:
209         print('Отношение является рефлексивным')
210     else:
211         print('Отношение не является рефлексивным')
212         flagR = False
213
214     print('\n')
215     if not flagS or not flagR or not flagT:
216         print('Так как отношение не обладает свойством ', end='')
217         if not flagS:
218             print('симметричности', end=', ')
219         if not flagT:
220             print('транзитивности', end=', ')
221         if not flagR:
222             print('рефлексивности', end=', ')
223         print('''то для получения фактор-множества отношения, требуется
224                 построить эквивалентное замыкание.''')
225
226         copy = matrix
```

```
227         ls, mt = make_equivalent_closure(copy, size)
228 
229         print('Эквивалентное замыкание бинарного отношения: {', end='')
230         print(*ls, sep=', ', end='} \n\n ')
231 
232         print('Матрица эквивалентного замыкания бинарного отношения:')
233         for i in range(len(mt)):
234             print(*mt[i])
235         print(' \n ')
236 
237         factor_set_res = factor_set(matrix, size)
238         print_factor(factor_set_res)
239         full_system_of_class_representatives(factor_set_res)
240 
241     else:
242         print('Заданное отношение является эквивалентным. Его матрица:')
243         for i in range(len(matrix)):
244             print(*matrix[i])
245         print(' \n ')
246 
247         factor_set_res = factor_set(matrix, size)
248         print_factor(factor_set_res)
249         full_system_of_class_representatives(factor_set_res)
250 
251 print('''Вы хотите получить минимальные/наименьшие и максимальные/наибольшие
252         элементы множества? Да (1) или Нет (0)''')
253 yes_or_no = int(input())
254 if yes_or_no:
255     print('Выберите тип задания множества: число (1) или заданное множество
        ↪ (2)')
256     set_type = int(input())
257 
258     print('Выберете тип порядка: <= (1) или отношение делимости (2)')
259     order_type = int(input())
260 
261     if set_type == 1:
262         print('Введите число')
263         num = int(input())
264         print('Хотите ли добавить единицу во множество? Да(1), Нет(0)')
265         yes_or_no = int(input())
266         sub_res = None
```

```
267         if yes_or_no == 1:
268             if order_type == 2:
269                 sub_res = dividers(num)
270                 res = hasse_division(sub_res)
271             else:
272                 sub_res = [i + 1 for i in range(num)]
273                 res = hasse_greater_eq(sub_res)
274         else:
275             if order_type == 2:
276                 sub_res = dividers(num, True)
277                 res = hasse_division(sub_res)
278             else:
279                 sub_res = [i + 2 for i in range(num - 1)]
280                 res = hasse_greater_eq(sub_res)
281     else:
282         print('Введите множество')
283         num = [int(value) for value in input().split()]
284         num = list(set(num))
285         num.sort()
286 
287         if order_type == 2:
288             res = hasse_division(num)
289         elif order_type == 1:
290             res = hasse_greater_eq(num)
291 
292     min_max_elements(res)
293     print('Вы хотите получить диаграмму Хассе? Да(1) или Нет(0)')
294     yes_or_no = int(input())
295     if yes_or_no:
296         import hasse_visualization as hv
297         if order_type == 1:
298             hv.visual(res, True)
299         else:
300             hv.visual(res)
301     print(res)
302 
303 '''
304 Примеры входных данных:
305 
306 3
307 0 1 0
```

```
0 0 1
1 0 0

4
1 0 1 0
1 1 0 0
0 0 1 0
0 1 0 1

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
'''
```

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.