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

```
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 бинарного отношения:
[{1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}, {1, 2, 3, 4}, {5}]
```

Рисунок 1

3.2 Код программы, реализующей рассмотренные алгоритмы

```
1  def make_set(matrix, size):
2
3      set_view = []
4
5      for i in range(size):
6          for j in range(size):
7              if matrix[i][j] == 1:
8                  set_view.append((i + 1, j + 1))
9      return sorted(set_view)
10
11
12 def matrix_set_view(matrix_set, flag=None):
13     if not flag:
14         print('Исходное отношение: {', end='')
15         print(*matrix_set, sep=', ', end='} \n ')
16     else:
17         print('{', end='')
18         print(*matrix_set, sep=', ', end='; ')
19
20
21 def print_factor(res_set, res_representatives, factor):
22
23     print('Фактор-множество отношения: {' + str(res_set)[1:-1] + '}, где ',
       ↪  end='')
```

```
24
25      for i, representative in enumerate(res_representatives):
26
27          print(str(res_set[i]) + ' \u2208 '
28                + f'\u03B5({representative})={list(map(lambda x: x[0],
                    ↪ factor[i]))}', end='')
29
30          if i < len(res_representatives) - 1:
31              print(', ', end='')
32          else:
33              print('\n')
34
35
36  def factor_set(matrix, size):
37      factor = [[(j + 1, i + 1) for j, value in enumerate(matrix[i]) if value ==
            ↪ 1] for i in range(size)]
38      factor.sort(key=len)
39
40      factor_res = []
41      factor_classes = []
42      for i in range(size):
43          for j in range(len(factor[i])):
44              if not (factor[i][j][0] in factor_res):
45                  factor_res.append(factor[i][j][0])
46                  factor_classes.append(factor[i][j][1])
47                  i += 1
48      return factor_res, factor_classes, factor
49
50
51  def make_equivalent_closure(copy, size):
52      list_for_equivalent_closure = set()
53      for i in range(size):
54          for j in range(size):
55              if matrix[i][j] == 1 and matrix[j][i] == 0:
56                  copy[j][i] = 1
57              if copy[j][i]:
58                  list_for_equivalent_closure.add((j + 1, i + 1))
59          if matrix[i][i] == 0:
60              copy[i][i] = 1
61          if copy[i][i]:
62              list_for_equivalent_closure.add((i + 1, i + 1))
```

```
63
64      for _ in range(size):
65          for k in range(size):
66              for i in range(size):
67                  for j in range(size):
68                      if copy[k][i] == copy[i][j] == 1 and copy[k][j] == 0:
69                          copy[k][j] = 1
70                      if copy[k][j]:
71                          list_for_equivalent_closure.add((k + 1, j + 1))
72
73      return sorted(list_for_equivalent_closure), copy
74
75
76  def is_transitive(matrix, size):
77
78      for k in range(size):
79          for i in range(size):
80              for j in range(size):
81                  if matrix[k][i] == matrix[i][j] == 1 and matrix[k][j] == 0:
82                      return False
83      return True
84
85
86  def is_symmetric_or_antisymmetric(matrix, size):
87
88      flag_symmetric = True
89      flag_antisymmetric = True
90
91      for i in range(size):
92          for j in range(size):
93              if not matrix[i][j] == matrix[j][i]:
94                  flag_symmetric = False
95              if matrix[i][j] == matrix[j][i] == 1 and not i == j:
96                  flag_antisymmetric = False
97              if not flag_symmetric and not flag_antisymmetric:
98                  return False, False
99
100     return flag_symmetric, flag_antisymmetric
101
102
103 def is_reflexive_or_anti_reflexive(matrix, size):
```

```
104
105     flag_reflexive = True
106     flag_anti_reflexive = True
107
108     for i in range(size):
109         if matrix[i][i] == 0:
110             flag_reflexive = False
111         elif matrix[i][i] == 1:
112             flag_anti_reflexive = False
113         if not flag_reflexive and not flag_anti_reflexive:
114             return False, False
115
116     return flag_reflexive, flag_anti_reflexive
117
118
119 def get_data():
120     n = int(input())
121     m = [[int(elem) for elem in input().split()] for _ in range(n)]
122     m_set = [(i + 1, j + 1) for i in range(n) for j in range(n) if m[i][j] ==
        ↪ 1]
123     return m, sorted(m_set), n
124
125
126 def hasse_greater_eq(nums):
127     res = []
128     res.append((nums[0], 1, []))
129     for i, num in enumerate(nums[1:]):
130         res.append((num, res[-1][1] + 1, [res[-1][0]]))
131     return res
132
133
134 def hasse_division(dividers_num):
135     hasse_list = []
136     sl = {key: 1 if key == 1 else 0 for i, key in enumerate(dividers_num)}
137
138     for number in dividers_num[1:]:
139         for divider in dividers_num[:dividers_num.index(number)]:
140             if number % divider == 0:
141                 sl[number] = sl[divider] + 1
142
143     for k, v in sl.items():
```

```
144         pod_res = []
145         for k1, v1 in sl.items():
146             if v1 + 1 == v and k % k1 == 0:
147                 pod_res.append(k1)
148         hasse_list.append((k, v, pod_res))
149     hasse_list.sort(key=lambda x: x[1])
150     return hasse_list
151 
152 
153 def dividers(num, flag=False):
154     result = []
155     begin = 1
156     if flag:
157         begin = 2
158     for i in range(begin, int(num / 2) + 1):
159         if num % i == 0:
160             result.append(i)
161     result.append(num)
162     return result
163 
164 ##########################################################################################
165 matrix, matrix_set, size = get_data()
166 matrix_set_view(matrix_set)
167 print('\n')
168 print('Свойства бинарного отношения:')
169 flagT = True
170 flagR = True
171 flagS = True
172 
173 if is_transitive(matrix, size):
174     print('Отношение является транзитивным')
175 else:
176     print('Отношение не является транзитивным')
177     flagT = False
178 
179 symm, _ = is_symmetric_or_antisymmetric(matrix,size)
180 if symm:
181     print('Отношение является симметричным')
182 else:
183     print('Отношение не является симметричным')
184     flagS = False
```

```
185
186 refl, _ = is_reflexive_or_anti_reflexive(matrix, size)
187 if refl:
188     print('Отношение является рефлексивным')
189 else:
190     print('Отношение не является рефлексивным')
191     flagR = False
192
193 print('\n')
194 if not flagS or not flagR or not flagT:
195     print('Так как отношение не обладает свойством ', end='')
196     if not flagS:
197         print('симметричности', end=', ')
198     if not flagT:
199         print('транзитивности', end=', ')
200     if not flagR:
201         print('рефлексивности', end=', ')
202     print('то для получения фактор-множества отношения, требуется построить
        ↪ эквивалентное замыкание.')
203
204     copy = matrix
205     ls, mt = make_equivalent_closure(copy, size)
206
207     print('Эквивалентное замыкание бинарного отношения: {', end='')
208     print(*ls, sep=', ', end='}\n\n')
209
210     print('Матрица эквивалентного замыкания бинарного отношения:')
211     for i in range(len(mt)):
212         print(*mt[i])
213     print('\n')
214
215     res_set, res_representatives, factor = factor_set(mt, size)
216     print_factor(res_set, res_representatives, factor)
217
218 else:
219     print('Заданное отношение является эквивалентным. Его матрица:')
220     for i in range(len(matrix)):
221         print(*matrix[i])
222     print('\n')
223     res_set, res_representatives, factor = factor_set(matrix, size)
224     print_factor(res_set, res_representatives, factor)
```

```
225 ##############################################################################
226
227 print('\u13AF\u212C\u2642\u0392\u20B3')
228 for _ in range(7):
229
        ↪ print('\u2694\u2694\u2694\u2694\u2694\u2694\u2694\u2694\u2694\u2694\u2694\u26
        ↪ end='')
230 print('\u2694\u2694\u2694\u2694\u2694\u2694\u2694\u2694\u2694\u2694\u2694\u2694\u2694
231
232
233 print('Выберите тип задания множества: число (1) или заданное множество (2)')
234 set_type = int(input())
235
236 print('Выбереете тип порядка: <= (1) или отношение делимости (2)')
237 order_type = int(input())
238
239
240 num = None
241 res = None
242 if set_type == 1:
243     print('Введите число')
244     num = int(input())
245     print('Хотите ли добавить единицу во множество? Да(1), Нет(0)')
246     yes_or_no = int(input())
247     sub_res = None
248     if yes_or_no == 1:
249         if order_type == 2:
250             sub_res = dividers(num)
251             res = hasse_division(sub_res)
252         else:
253             sub_res = [i + 1 for i in range(num)]
254             res = hasse_greater_eq(sub_res)
255     else:
256         if order_type == 2:
257             sub_res = dividers(num, True)
258             res = hasse_division(sub_res)
259         else:
260             sub_res = [i + 2 for i in range(num - 1)]
261             res = hasse_greater_eq(sub_res)
262 else:
263     print('Введите множество')
```

```
264     num = [int(value) for value in input().split()]
265     num.sort()
266     if order_type == 2:
267         res = hasse_division(num)
268     elif order_type == 1:
269         res = hasse_greater_eq(num)
270 
271 print(res)
272 
273 print('Вы хотите получить диаграмму Хассе? Да(1) или Нет(0)')
274 yes_or_no = int(input())
275 if yes_or_no:
276     # import ass
277     # ass.main(res)
278 '''
279 Примеры входных данных:
280 
281 3
282 0 1 0
283 0 0 1
284 1 0 0
285 
286 4
287 0 1 1 0
288 1 1 1 0
289 0 1 1 0
290 0 0 0 1
291 
292 4
293 0 1 0 0
294 0 0 0 0
295 0 0 0 1
296 0 1 0 0
297 
298 4
299 1 1 0 1
300 0 1 1 0
301 0 0 1 1
302 0 0 0 1
303 
304 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
'''
```

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.