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

КЛАССИФИКАЦИЯ БИНАРНЫХ ОТНОШЕНИЙ И СИСТЕМЫ ЗАМЫКАНИЙ

ЛАБОРАТОРНАЯ РАБОТА

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

Проверил
аспирант _____ В. Н. Кутин

Саратов 2022

СОДЕРЖАНИЕ

1 Постановка задачи

Цель работы:

Изучение основных свойств бинарных отношений и операций замыкания бинарных отношений.

Порядок выполнения работы:

1. Рассмотреть понятия полугруппы, подполугруппы и порождающего множества. Разработать алгоритм построения подполугрупп по по таблице Кэли.
2. Разработать алгоритм построения полугруппы бинарных отношений по заданному порождающему множеству.
3. Рассмотреть понятия подгруппы, порождающего множества и определяющих соотношений. Разработать алгоритм построения полугруппы по порождающему множеству и определяющим соотношениям.

2 Теоретические сведения по рассмотренным темам с их обоснованием

Полугруппа – это алгебра $S = (S, \cdot)$ с однойассоциативной бинарной операцией $\cdot$, т.е. выполняется $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ для любых $x, y, z \in S$.

Если полугрупповая операция называется умножением (соответственно, сложением), то полугруппу называют мультипликативной (соответственно, аддитивной).

Подмножество X полугруппы S называется **подполугруппой**, если X устойчиво относительно операции умножения, т.е. для любых $x, y \in X$ выполняется свойство: $x \cdot y \in X$.

В этом случае множество X с ограничением на нем операции умножения исходной полугруппы S образует полугруппу.

В силу общего свойства подалгебр пересечение любого семейства X_i $(i \in I)$ подполугрупп полугруппы S является подполугруппой S и, значит, множество $Sub(S)$ всех подполугрупп полугруппы S является системой замыканий. множество X. Такая полугруппа обозначается символом $\langle X \rangle$ и называется подполугруппой S, порождённой множеством X. При этом множество X называется также **порождающим множеством** подполугруппы $\langle X \rangle$. В частности, если $\langle X \rangle = S$, то X называется порождающим множеством полугруппы S и говорят, что множество X порождает полугруппу S.

Для любой конечной полугруппы S найдется такой конечный алфавит A, что для некоторого отображения $\phi : A \to S$ выполняется равенство $< \phi(A) >= S$ и, значит, $S \cong A^+/ker\phi$ этом случае множество A называется множеством порождающих символов полугруппы S (относительно отображения $\phi : A \to S$). Если при этом для слов $w_1, w_2 \in A$ выполняется равенство $\phi(w_1) = \phi(w_2)$, т.е. $w_1 \equiv w_2(ker\phi)$, то говорят, что на S выполняется соотношение $w_1 = w_2$ (относительно отображения $\phi : A \to S$).

Очевидно, что в общем случае множество таких соотношений $w_1 = w_2$ для всех пар $(w_1, w_2) \in ker\phi$ будет бесконечным и не представляется возможности эффективно описать полугруппу S в виде полугруппы классов конгруэнции $ker\phi$. Однако в некоторых случаях можно выбрать такое сравнительно простое подмножество $\rho \subset ker\phi$, которое однозначно определяет конгруэнцию $ker\phi$ как наименьшую конгруэнцию полугруппы A^+ , содержащую отношение ρ, т.е. $ker\phi = f_{con}(\rho) = f_{eq}(f_{reg}(\rho))$.

Так как в случае $(w_1, w_2) \in \rho$ по-прежнему выполняется равенство $\phi(w_1) = \phi(w_2)$, то будем писать $w_1 = w_2$ и называть такие выражения **определяющими соотношениями**. Из таких соотношений конгруэнция $ker\phi$ строится с помощью применения следующих процедур к словам $u, v \in A^+$:

(а) слово v непосредственно выводится из слова u, если v получается из u заменой некоторого подслова w_1 на слово w_2, удовлетворяющее определяющему соотношению $w_1 = w_2$, т.е. $(u, v) = (xw_1y, xw_2y)$ для некоторых $x, y \in A^*$;

(б) слово v выводится из слова u, если v получается из u с помощью конечного числа применения процедуры (а).

Если все выполняющиеся на S соотношения выводятся из определяющих соотношений совокупности ρ, то конгруэнция $ker\phi$ полностью определяется отношением ρ и выражение $< A : w_1 = w_2 : (w_1, w_2) \in \rho >$ называется **копредставлением полугруппы** S.

3 Результаты работы

3.1 Алгоритм 1 - Построение подполугруппы по заданному порождающему множеству

Вход: Полугруппа S с таблицей Кэли $A = (a_{ij})$ размерности $n \times n$ и подмножество $X \subset S$.

Выход: Подполугруппа $\langle X \rangle \subset S$.

Шаг 1. Положим $i = 0$, $X_0 = X$.

Шаг 2. Для X_i вычислим $\overline{X}_l = \{x \cdot y : x \in X_i \wedge y \in X\}$ и положим $X_{i+1} = X_i \cup \overline{X}_l$ (выражение $x \cdot y$ означает a_{xy} в таблице Кэли A).

Шаг 3. Если $X_{i+1} = X_i$ вернем X_i, которое будет являться подполугруппой $\langle X \rangle \subset S$, иначе положим $i = i + 1$ и вернемся ко 2-му шагу.

Трудоемкость алгоритма $O(n^3)$.

3.2 Коды программ, реализующей рассмотренные алгоритмы

```
import numpy as np
from itertools import product

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


def subsets(n):
    nums = [i for i in range(n)]
    from functools import reduce
    return reduce(lambda res, x: res + [subset + [x] for subset in res],
                  nums, [[]])[1:]


def get_res(matrixes):
    res = []
    for subset in subsets(len(matrixes)):
        podres = matrixes[subset[0]]
        for index in subset[1:]:
            podres *= matrixes[index]
```

```
        for relation in get_set(podres):
            res.append(relation)
    return set(res)


def add_correlation(cur_word, alph, dct, semigroup_elems):
    new_words = []
    for letter in alph:
        new_word = cur_word + letter
        m = []
        for i in dct[cur_word]:
            if i != '*':
                m.append(dct[letter][semigroup_elems.index(i)])
            else:
                m.append('*')
        flag = True
        for key in dct:
            if m == dct[key]:
                print(new_word, '->', key)
                flag = False
        if flag:
            dct[new_word] = m
            new_words.append(new_word)
    return new_words


def find_correlation(ans):
    result = {}
    correlations = {}
    for key, value in ans.items():
        if not any(np.array_equal(value, i) for i in result.values()):
            result[key] = value
        else:
            for k, v in result.items():
                if np.array_equal(v, value):
                    correlations[key] = k
    print("Coopresentation: ")
    for key, value in result.items():
        print(key, ":\n", value)

    print("The resulting ratios: ")
```

```
    for key, value in correlations.items():
        print(key, "->", value)


def task1():
    print("Enter your set")
    st = list(input().split())
    print('Enter Cayley table')
    print("  ", *st)
    matrix = []
    for i in range(len(st)):
        print(st[i], end="  ")
        s = list(input().split())
        matrix.append(s)
    print("Enter your subset")
    subst = list(input().split())
    x_i = subst.copy()
    while True:
        x_l = []
        for x in x_i:
            for y in subst:
                x_l.append(matrix[x_i.index(x)][st.index(y)])
        x_0 = x_i.copy()
        x_0.sort()
        x_i = list(set(x_i).union(set(x_l)))
        x_i.sort()
        if x_0 == x_i:
            print('Subsemigroup is', *x_i)
            break


def task2():
    print("Enter the elements of the set: ")
    input_list = input().replace(",", "").split()
    n = len(input_list)
    print("Enter the number of binary relations")
    bin_relation_amount = int(input())
    bin_relation_matrices = {}
    for i in range(1, bin_relation_amount + 1):
        print(f"Enter boolean matrix Values {i} binary relation: ")
        print(" ", *input_list)
```

```
        matrix = [list(map(int, input(f"{input_list[i]} ").split())) for i in range(n)
        matrix = np.array(matrix).reshape(n, n)
        bin_relation_matrices[str(i)] = matrix

    combinations_list = []
    for i in range(1, bin_relation_amount + 1):
        combinations = list(product(''.join([str(elem) for elem in range(1, bin_relati
        combinations_list += combinations

    for comb in combinations_list:
        cur_matrix = bin_relation_matrices[comb[0]].copy()
        word = comb[0]
        for comb_i in range(1, len(comb)):
            cur_matrix *= bin_relation_matrices[comb[comb_i]]
            word += comb[comb_i]
        bin_relation_matrices[word] = cur_matrix

    find_correlation(bin_relation_matrices)


def task3():
    print("Enter semigroup elements: ")
    semigroup_elems = [elem for elem in input().replace(",", "").split()]
    n = len(semigroup_elems)
    print("Enter the elements of the transformation set: ")
    generators_list = input().replace(",", "").split()
    gn = len(generators_list)
    translation_dict = {}
    for i in range(gn):
        print(f"Enter transformation values '{generators_list[i]}' elements of the ser
        print((str(semigroup_elems)[1:-1]).replace(", ", " ").replace("'", ""))
        translation = input().split()
        translation_dict[generators_list[i]] = translation
    print("Coopresentation: ")
    gl = generators_list.copy()
    while gl:
        gl1 = []
        for s in gl:
            gl1.extend(add_correlation(s, generators_list, translation_dict, semigroup
        gl = gl1
    print("The resulting ratios: ")
```

```
    print(translation_dict)

task3()
```

3.3 Результаты тестирования программ

```
How you want enter your relation (set or matrix)?
set
Enter the number of elements in relation
3
Enter your set
(1,2) (1,3)
----------------------------------------
Relation's matrix
[[0 1 1]
 [0 0 0]
 [0 0 0]]
----------------------------------------
Set is anti-reflexive
Set is anti-symmetry
Set is transitive
----------------------------------------
Relation is the relation of the strict order
----------------------------------------
Closure matrix:
[[1 1 1]
 [1 1 0]
 [1 0 1]]
----------------------------------------
Closure set:
(1, 1) (1, 2) (1, 3) (2, 1) (2, 2) (3, 1) (3, 3)
----------------------------------------
```

Рисунок 1 – Ввод бинарного отношения (1, 2), (1, 3) в виде множества с выводом свойств этого множества и замыкания относительно эквивалентности

3.4 Оценки сложности рассмотренных алгоритмов

3.4.1 Алгоритм определения рефлексивности

Сложность выполнения проверки на рефлексивность определяется как $O(n)$.

3.4.2 Алгоритм определения антирефлексивности

За счет схожести с алгоритмом определения рефлексивности сложность определяется как $O(n)$.

3.4.3 Алгоритм определения симметричности

Сложность транспонирования в numpy определяется как $O(n^{3/2}log\ n)$, сложность сравнение двух матриц поэлементно определяется как $O(n^2)$. Отсюда можно сделать вывод, что сложность алгоритма будет определятся как $O(n^{3/2}log\ n + n^2) = O(n^2)$.

3.4.4 Алгоритм определения антисимметричности

Сложность поэлементного умножения матриц определяется как $O(n^2)$, сложность сравнение двух матриц поэлементно определяется как $O(n^2)$. Отсюда можно сделать вывод, что сложность будет определятся как $O(n^2 + n^2) = O(n^2)$

3.4.5 Алгоритм определения транзитивности

Из всего выше сказанного очевидно, что сложность проверки на транзитивность или антитранзитивность составляет $O(n^3)$, так как в нем используется умножение, сравнение матриц и тройной цикл для проверки рефлексивности в худшем случае матриц.

3.4.6 Алгоритм классификации

Сложность выполнения самого алгоритма классификации бинарных отношений реализованно через словарь языка python и оператор if, поэтому является константной ($O(1)$), если не учитывать сложность выполнения проверки свойств отношения. Если учитывать сложность алгоритмов проверки свойств отношения, то :

1. Сложность проверка на квазипорядок определяется как $O(n^3 + n) = O(n^3)$.
2. Сложность проверка на эквивалентность определяется как $O(n^3 + n + n^{3/2}log\ n) = O(n^3)$.

3. Сложность проверка на частичный порядок определяется как $O(n^3 + n + n^3) = O(n^3)$.
4. Сложность проверка на строгий порядок определяется как $O(n^3 + n + n^3) = O(n^3)$.

3.4.7 Построение замыкания рефлексивности

Так как весь алгоритм строится на заполнении главной диагонали матрицы 1, то его сложность состовляет $O(n)$.

3.4.8 Построение замыкания симметричности

Для посторения замыкания симметричности используются вложенный цикла, поэтому сложность алгоритма определяется как $O(n^2)$.

3.4.9 Построение замыкания транзитивности

Для посторения замыкания транзитивности используются три вложенный цикла, поэтому сложность алгоритма определяется как $O(n^4)$.

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы были рассмотренны теоритические основы свойств бинарных отношений, их видов и методов их замыкания по каждому из свойств. На основе этой теоретической части была смоделирована программа на языке Python с использованием средств библиотеки Numpy, которая способна определить свойства заданного множества, его вид и построить систему замыкания по каждому из основных свойств бинарного отношения, а так же была оценена асимптотика каждого реализованного алгоритма.