Московский Государственный Университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математической кибернетики

КУРСОВАЯ РАБОТА

Тема курсовой работы:

«Возможность доопределения частичной трёхзначной функции»

Выполнил: Смехов А. Д.
Студент 518/1 группы

Научный руководитель: Нагорный А. С.
старший преподаватель, к. ф.-м. н.

Москва, 2021

Содержание

Введение

Функции многозначной логики – один из основных модельных объектов дискретной математики, широко используемый для описания разнообразных дискретных систем. Классификация множеств P_k функций k-значной логики на основе оператора суперпозиции, являются самыми известными и самыми изученными. Однако, в отличие от случая булевых функций $(k = 2)$, при $k \geq 3$ результаты менее исчерпывающие, чем для множества P_2, полученные Э. Постом.

В данной работе рассматриваются условия принадлежности функции каждому предполному классу в P_3. Кроме того, приводится их реализация на языке Python для определения принадлежности частично заданной функций предполному классу или объединению предполных классов.

1.1 Постановка задачи

Пусть $K_1, K_2, \dots, K_N$ – предполные классы в P_3, $f(\alpha_1, \alpha_2, \dots, \alpha_m)$ – частично заданная функция от m переменных, где $\alpha_i \subseteq \{0, 1, 2, -\}$. Стоит задача проверки функции f на принадлежность классам $K_1, K_2, \dots, K_N$, и их объединению. А также задача доопределения функции f до принадлежащей классу K_i путём замены $\{-\}$ на $\{0, 1, 2\}$ в случае, если невозможно определить принадлежит ли функция классу K_i или нет.

1.2 Теоретическая часть

Как известно [4], в P_3 существует ровно 18 предполных классов:

T_0, T_1, T_2 – классы функций, сохраняющих константу 0, 1, 2 соответственно.

T_{12}, T_{01}, T_{02} – классы функций, сохраняющих константу множество $\{1, 2\}$, $\{0, 1\}$, $\{0, 2\}$ соответственно.

$B -$ класс Слупецкого

$S -$ класс функций, самодвойственных относительно перестановки (120).

M_0, M_1, M_2 – классы функций, монотонных относительно порядка $2 < 0 < 1, 0 < 1 < 2, 1 < 2 < 0$ соответственно.

U_0, U_1, U_2 – классы функций, сохраняющих разбиение $\{\{0\},\{1,2\}\},\{\{1\},\{2,0\}\},\{\{2\},\{0,2\}\}$ соответственно.

C_0, C_1, C_2 – классы функций, сохраняющих 2-местный предикат $((x = y V \sigma \in \{x, y\}$ для любого $\sigma \in E3$.

L – класс линейных функций

Каждый из выше перечисленных классов является предикатно описуемым:

$$T_0 = Pol_3(0) \qquad T_1 = Pol_3(1) \qquad T_2 = Pol_3(2)$$

$$T_{12} = Pol_3(12) \qquad T_{01} = Pol_3(01) \qquad T_{02} = Pol_3(02)$$

$$B = Pol_3\left(\bigvee_{1 \le i < j \le 3} (x_i = x_j)\right) \qquad S = Pol_3\binom{012}{120}$$

$$M_0 = Pol_3\binom{001112}{021022} \qquad M_1 = Pol_3\binom{000112}{012122} \qquad M_2 = Pol_3\binom{000122}{012112}$$

$$U_0 = Pol_3\binom{01122}{01212} \qquad U_1 = Pol_3\binom{00122}{02102} \qquad U_2 = Pol_3\binom{00112}{01012}$$

$$C_0 = Pol_3\binom{0120012}{0121200} \qquad C_1 = Pol_3\binom{0121102}{0120211} \qquad C_2 = Pol_3\binom{0122201}{0120122}$$

$$L = Pol_3\begin{pmatrix} 000000000111111111222222222 \\ 000111222000111222000111222 \\ 012012012012012012012012012 \\ 021102210102210021210021102 \end{pmatrix}$$

Условия принадлежности функции предполному классу

Прежде, чем приступить к решению поставленных задач, необходимо сформулировать условия проверки функции на принадлежности каждому из классов. Но для начала нужно ввести ряд обозначений:

$Im(f) = \{\dots\} -$ множество значений функции f.

$Im(f, A) = \{\dots\} -$ множество значений функции f на множестве A.

Далее представлены условия проверки функции на принадлежность каждому предполному классу.

Утверждение: $f \in T_E \Leftrightarrow Im(f, E) \subseteq E$, где

$$E = \{0\}, \{1\}, \{2\}, \{0,1\}, \{0,2\}, \{1,2\}$$

Утверждение: $f \in S \Leftrightarrow \begin{cases} f(x_1, \dots, x_n) = \sigma \\ f(y_1 = x_1 + 1, \dots, y_n = x_n + 1) = \sigma + 1 \\ f(z_1 = x_1 + 2, \dots, z_n = x_n + 2) = \sigma + 2 \end{cases}$, где

$$x_i, y_i, z_i, \sigma \in E_3 \text{ при } i = \overline{1,n}$$

Утверждение: $f \in B \Leftrightarrow \begin{bmatrix} Im(f) = \{a, b\}, a \neq b \\ f(x_1, \dots, x_n) = f(x_i) \end{bmatrix}$

Утверждение: $f \in M_E \Leftrightarrow f(\alpha) \leq f(\beta) \leq f(\gamma)$, при $\alpha \leq \beta \leq \gamma$, где

$$E = \{0\}, \{1\}, \{2\} - \text{ограниченный частичный порядок}$$

Утверждение: $f \in L \Leftrightarrow f(\alpha_1, \alpha_2, \dots, \alpha_n) =$

$$= f(0, \dots, 0) + \big(f(\alpha_1, 0, \dots, 0) - f(0, \dots, 0)\big) + \dots + \big(f(0, 0, \dots, \alpha_n) - f(0, \dots, 0)\big)$$

, где $\alpha_i \in \{1, 2\}$, при чём $f(0, \dots, 2, \dots, 0) = 2 * f(0, \dots, 1, \dots, 0) - f(0, \dots, 0)$

, при $i = \overline{1,n}$

Утверждение: $f \in U_0 \Leftrightarrow \begin{bmatrix} Im\big(f(\beta)\big) = \{1, 2\} \\ f(\beta) \equiv 0 \end{bmatrix}$, где $\beta = (\beta_1,\ \beta_2, \dots, \beta_n)$,

$\beta_j = \begin{cases} 0 \text{ при } j = i_l (1, 2, \dots, s) \\ \neq 0 \text{ иначе} \end{cases}$ при любых $1 \leq i_1 \leq i_2 \leq \dots \leq i_s \leq n$

Утверждение: $f \in U_1 \Leftrightarrow \begin{bmatrix} Im\big(f(\beta)\big) = \{0, 2\} \\ f(\beta) \equiv 1 \end{bmatrix}$, где $\beta = (\beta_1,\ \beta_2, \dots, \beta_n)$,

$\beta_j = \begin{cases} 1 \text{ при } j = i_l (1, 2, \dots, s) \\ \neq 1 \text{ иначе} \end{cases}$ при любых $1 \leq i_1 \leq i_2 \leq \dots \leq i_s \leq n$

Утверждение: $f \in U_2 \Leftrightarrow \begin{bmatrix} Im\big(f(\beta)\big) = \{0,1\} \\ f(\beta) \equiv 2 \end{bmatrix}$, где $\beta = (\beta_1,\ \beta_2, \dots, \beta_n)$,

$\beta_j = \begin{cases} 2 \text{ при } j = i_l (1,2,\dots,s) \\ \neq 2 \text{ иначе} \end{cases}$ при любых $1 \le i_1 \le i_2 \le \cdots \le i_s \le n$

Утверждение: $f \in \mathrm{C}_0 \Leftrightarrow f(\beta) \neq f(\alpha)$, при $\beta = (\beta_1,\ \beta_2, \dots, \beta_n)$:

$\beta_i \neq \alpha_i$, где $\alpha_i = \{1\ ,2\}, Im\big(f(\alpha)\big) = \{1,2\}$.

Утверждение: $f \in \mathrm{C}_1 \Leftrightarrow f(\beta) \neq f(\alpha)$, при $\beta = (\beta_1,\ \beta_2, \dots, \beta_n)$:

$\beta_i \neq \alpha_i$, где $\alpha_i = \{0,2\}, Im\big(f(\alpha)\big) = \{0,2\}$.

Утверждение: $f \in \mathrm{C}_2 \Leftrightarrow f(\beta) \neq f(\alpha)$, при $\beta = (\beta_1,\ \beta_2, \dots, \beta_n)$:

$\beta_i \neq \alpha_i$, где $\alpha_i = \{0,1\}, Im\big(f(\alpha)\big) = \{0,1\}$.

Реализация доопределения частичной трехзначной функции

В ходе работы была реализована программа на языке программирования Python, которая принимает на вход вектор значений частично заданной функции и список предполных классов, на принадлежность которым надо проверить заданную функцию.

Программа состоит из 8 основных функций (функция для классов T_0, T_1, T_2; функция для классов T_{12}, T_{01}, T_{02}; функция для класса B; функция для класса S, функция для класса L; функция для классов M_0, M_1, M_2; функция для классов U_0, U_1, U_2; функция для классов $\mathrm{C}_0, \mathrm{C}_1, \mathrm{C}_2$) и 8 вспомогательных.

Программа способна работать в трёх режимах:

1. проверка на принадлежность функции классу/пересечению/объединению классов,
2. подсчёт возможных векторов, принадлежащих классу/пересечению/объединению классов, на основе исходного вектора,
3. построение возможных векторов, принадлежащих классу/пересечению/объединению классов, на основе исходного вектора.

Принадлежность классам производится в соответствии с утверждениями из предыдущего раздела.

Доопределение функции производится путём рассмотрения возможных (в соответствии с утверждениями) вариантов.

В случаях, когда функция гарантировано не принадлежит классу, попыток доопределения не производится.

В случаях с группами классов M_0, M_1, M_2; U_0, U_1, U_2; $\mathrm{C}_0, \mathrm{C}_1, \mathrm{C}_2$ вектор значений сначала приводится к эквивалентному вектору M_1, U_0, C_0 соответственно. После проверки на принадлежность и доопределения функции, результирующие вектора приводятся к изначальному классу.

При вызове функции для M_0, M_1, M_2 значения вектора функции заменяются на 4, 5, 6 в соотвествие с частичным порядком класса. По завершению работы значению обратно переводятся в пространство E_3.

3.1 Алгоритмы проверки вектора на принадлежность классу

В данном подразделе описывается алгоритмы, использованные в реализации проверки вектора значений функции на принадлежность классу.

Классы T_0, T_1, T_2: Для проверки функции на принадлежность классу T_0, T_1, T_2 проверяется в зависимости от класса значение на наборе $(000), (111), (222)$ на равенство $0, 1, 2$ соотвественно.

Класс B: В первую очередь функция проверяется на количество существенных переменных, если их количество равняется одному, то проверка заканчивается с положительным результатом (функция принадлежит классу). В противном случае полностью проходится вектор значений, и запоминаются возможные значения в векторе, как только программа встречает все три возможных значения $(0, 1, 2)$ в векторе функции, то проверка завершается с отрицательным результатом (функция не принадлежит классу). Если же после просмотра всего вектора количество возможных значений не превышает двух, то программа выдаёт положительный результат.

Классы T_{12}, T_{01}, T_{02}: Данная проверка реализована рекурсивным алгоритмом. Вектор значений делится на три равные части. Далее выбираются в зависимости от класса T_{12}, T_{01}, T_{02} вторая и третья часть, первая и вторая, первая и третья части. Выбраные части заново подаются на вход программе. Если длина полученного программой вектора равняется трём, то вектор также делится на три части и две соотвествующие классу части проверяются на допустимые значения: $T_{12}: \{1, 2\}, T_{01}: \{0, 1\}, T_{02}: \{0, 2\}$. Если все финальные вектора длины три прошли проверку на допустимые значения, то программа возвращает положительный результат, иначе отрицательный.

Класс S: Вектор значений разбивается на три равные части. Далее для каждого значения из первой части находятся соотвествующие ему значения из второй и третьих частей (Например, для значения на наборе (012), соотвествующие ему значения из второй и третьей частей лежат на наборах (120) и (201) соотвественно). Затем эти значения проверяются на условие самодвойственности, то есть если значения a_1, a_2, a_3 лежат в первом, втором и третьем наборе соотвественно, то они связаны следующими соотношениями: $a_2 \equiv a_1 + 1(mod\ 3)$, $a_3 \equiv a_2 + 1(mod\ 3)$. Проверка заканчивается либо отрицательным результатом при первом невыполнение условия самодвойственности, либо положительным результатом, если проверены все значение из первой части вектора.

Классы M_0, M_1, M_2: В первую очередь вектор значений рекурсивным алгоритмом приводится к частичному порядку $0 < 1 < 2$: вектор делится на три части, части меняются местами в соотвествующем порядке, предыдущие действия выполняются для каждой части. Затем в соотвествие с изначальным частичным порядком наименьшие значения в векторе меняются на 4, наибольшие на 6, средние на 5. Далее выполняется сама проверка: вектор делится на три части, каждое значение из первой части сравнивается с тем же по счёту значением из второй части, каждое значение из второй части сравнивается с тем же по счёту значением из третьей части. В случае, если значение из первой части больше значения из второй или значение из второй части больше значения из третьей части, то проверка завершается с отрицательным результатом. Иначе алгоритм проверка выполняется для каждой из частей отдельно. Если все проверки прошли успешно, то программа возвращает положительный результат.

Классы U_0, U_1, U_2: В первую очередь вектор значений рекурсивным алгоритмом приводится к единому для всех классов виду: вектор делится на три части, часть с номером класса ставится на первое место, две другие в том же порядке занимают первое и второе место. Затем описанные действия совершаются над каждой из частей отдельно. Сама проверка также

реализована рекурсивным алгоритмом: вектор единого вида делится на три части, далее каждое значение из второй части сравнивается со значением с тем же номером из третьей части. Если эти значения не равны между собой и хотя бы одно из них равно номеру класса, то программа возвращает отрицательный результат. Иначе проверка проводится для каждой из частей отдельно. Если все проверки прошли успешно, то программа возвращает положительный результат.

Классы $\mathrm{C}_0, \mathrm{C}_1, \mathrm{C}_2$: Точно таким же способом, как и в случае с классами U_0, U_1, U_2, вектор значений приводится к единому для всех классов виду. Сама проверка также реализована аналогичным рекурсивным алгоритмом. Отличие заключается в условие непринадлежности: если значение из первой части не равно значению с темже номером из второй или третьей части, и ни одно из этих значений не равно номеру класса, или значение из первой части не равно первому значению из второй или третьей части, и ни одно из них не равно номеру класса, то программа возвращает отрицательный результат.

Класс L: Запоминается значение на наборе, состоящем из всех нулей, а также на наборах содержащих одну единицу. Далее проверяются наборы, содержащие ровно одну двойку, они должны быть равны удвоенному значению, соотвестующего набора с одной единицой, с вычитаением значения нулевого набора. Далее все оставшиеся наборы представляются как сумма наборов, содержащих одну единицы или двойку, и проверяются на соотвествие значения набора полученной сумме значений наборов. Если все проверки прошли успешно, то программа выдаёт положительный резльтат, иначе отрицательный.

3.2 Примеры использования программы

Ниже представлены ряд примеров работы программы. Для обозначения неопредёленных значений функции используется цифра 3.

В первом примере функция проверяется на принадлежность пересечению классов T0, T1, T2, затем программа вызывается повторно, но уже с целью подсчёта возможных векторов, принадлежащих пересечению.

	Пример 1.
Вход	033313332 T0,T1,T2
Выход	Функция [0 3 3 3 1 3 3 3 2] лежит в пересечение классов ['T0' 'T1' 'T2']
	Из вектора значений [0 3 3 3 1 3 3 3 2] можно получить 729 функций, лежащих в пересечение классов ['T0' 'T1' 'T2']

Во втором примере на вход подаётся функция заведомо не лежащая, в классе S. Программа вызывается два раза: в первом случае производится проверка на принадлежность пересечению входных классов, во втором проверяется принадлежность каждому из входных классов.

	Пример 2.
Вход	003123333 S,T0,T2
Выход	Функция [0 0 3 1 2 3 3 3 3] не лежит в объединение классов ['S' 'T0' 'T2']
	Принадлежность классу: Функция [0 0 3 1 2 3 3 3 3] не лежит в классе S Функция [0 0 3 1 2 3 3 3 3] лежит в классе T0 Функцию [0 0 3 1 2 3 3 3 3] можно дополнить до принадлежности классу T2

В третьем примере программа получается на вход полностью неопределённую функцию одной переменной. В качестве результата

программа выводит все возможные векторы, принадлежащие данному пересечению классов.

Пример 3.	
Вход	333 M0,U0,L
Выход	Из вектора значений [3 3 3] можно получить следующие функции, лежащие в пересечение классов ['M0' 'U0' 'L']: [2 2 2] [0 1 2] [0 0 0] [1 1 1]

В четвёртом примере программа проверяет вектор на принадлежность классу, а при повторном запуске строит все возможные векторы, принадлежащие классу, на основе исходного вектора.

Пример 4.	
Вход	133 C0
Выход	Функцию [1 3 3] можно дополнить до принадлежности классу C0
	Из вектора значений [1 3 3] можно получить следующие функции, лежащие в классе C0: [1 0 0] [1 0 1] [1 1 0] [1 1 1]

В пятом примере программа проверяет вектор на принадлежность объединению классов и выводит все возможные векторы, принадлежащие данному объединению классов.

Пример 5.	
Вход	133 T0,C0,C1,C2
Выход	Функцию [1 3 3] можно дополнить до принадлежности объединению классов ['T0' 'C0' 'C1' 'C2']
	Из вектора значений [1 3 3] можно получить следующие функции, лежащие в объединении классов ['T0' 'C0' 'C1' 'C2']: [1 0 0] [1 0 1] [1 1 0] [1 1 1] [1 1 2] [1 2 1] [1 2 2] [1 0 2]

Заключение

В процессе выполнения работы:

- Были сформулированы условия принадлежности трёхзначной функции предполным классам P_3.
- Была разработана программа, проверяющая частично заданную трёхзначную функцию на принадлежность предполному классу или объединению предполных классов, а так же строящая возможные вектора функций, принадлежащих заданному классу или объединению классов, на основе исходного частично заданного вектора функции.

Список литературы

1. Жук Д. Н. От двухзначной к k-значной логике // Интеллектуальные системы. Теория и приложения, 2018, том 22, выпуск 1, С. 131–149.
2. Нагорный А. С. О свойствах предполных классов в P3 // Известия высших учебных заведений. Поволжский регион. Физ.-мат. науки. Пенза: Изд-во Пензенского государственного университета, 2012, №2 (22), С. 16-24.
3. Нагорный А. С. О пересечениях и объединениях предполных классов многозначной логики: канд. дис: 01.01.09// МГУ имени М. В. Ломоносова, Москва, 2013, 260 с.
4. Яблонский С. В. Функциональные построения в k-значной логике // Тр. МИАН СССР им. В. А. Стеклова. 1958. 51. С. 5–142.

Приложение

Исходный код программы

```
#!/usr/bin/env python
# coding: utf-8

import numpy as np
import math
import itertools
import random
import os
import sys


def read_input(input_path):
    input_vector = []
    classes = []

    input_exist = os.path.exists(input_path)
    if input_exist:
        with open(input_path, 'r') as file:
            i = 0
            for line in file:
                if i == 0:
                    for var in range(len(line) - 1):
                        input_vector.append(int(line[var]))
                    i += 1
                elif i == 1:
                    for var in line.split(','):
                        classes.append(var)
                    i += 1

        input_vector = np.array(input_vector, dtype = np.int32)
        classes = np.array(classes)

    return input_vector, classes




#Проверка на T0, T1, T2
"""
Вход:
vector - вектор значений
check_type - тип проверки:
    0:
    Проверяет принадлежит ли классу вектор
    Выход:
        0 - не принадлежит
        1 - принадлежит
        2 - можно дополнить

    1:
    Если вектор принадлежит классу, то выдаёт количество возможных векторов
    Выход:
        0 - не принадлежит
        1 и больше - количество возможных векторов
    2:
    Если вектор принадлежит классу, то выдаёт возможные вектора
    Выход:
        Массив возможных векторов
T_type - 0: T0, 1: T1, 2: T2
"""
def T0_T1_T2_check(vector, check_type, T_type):
    check_pos, correct_digit, wrong_digit_1, wrong_digit_2 = 0, 0, 1, 2
```

```
    if T_type == 1:
        check_pos, correct_digit, wrong_digit_1, wrong_digit_2 = int(len(vector) / 2),
1, 0, 2
    elif T_type == 2:
        check_pos, correct_digit, wrong_digit_1, wrong_digit_2 = len(vector) - 1, 2,
0, 1

    if check_type == 0:
        if vector[check_pos] == 3:
            return 2
        elif vector[check_pos] != correct_digit:
            return 0
        else:
            return 1

    elif check_type == 1:
        add_amount = 0
        if   (vector[check_pos]   !=   wrong_digit_1)   and   (vector[check_pos]   !=
wrong_digit_2):
            add_amount += 1
            for i in range(len(vector)):
                if (vector[i] == 3) and (i != check_pos):
                    add_amount *= 3
        return add_amount

    elif check_type == 2:
        poss_vectors = []

        if   (vector[check_pos]   !=   wrong_digit_1)   and   (vector[check_pos]   !=
wrong_digit_2):
            poss_vectors.append(np.copy(vector))

            for i in range(len(vector)):
                if i == check_pos and vector[i] == 3:
                    for temp in poss_vectors:
                        temp[i] = correct_digit
                elif vector[i] == 3:
                    new_vectors = []
                    for temp in poss_vectors:
                        temp[i] = 0
                        new_vectors.append(np.copy(temp))

                        temp[i] = 1
                        new_vectors.append(np.copy(temp))

                        temp[i] = 2
                        new_vectors.append(np.copy(temp))

                    poss_vectors = new_vectors

        poss_vectors = np.array(poss_vectors)
        return poss_vectors




#Построение векторов из двух значений
"""
Вход:
vector - вектор значений
a - первое значение
b - второе значение
"""
def build_vectors(vector, a, b):
    poss_vectors = []
    poss_vectors.append(np.copy(vector))

    for i in range(len(vector)):
```

```
        if vector[i] == 3:
            new_vectors = []
            for temp in poss_vectors:
                temp[i] = a
                new_vectors.append(np.copy(temp))

                temp[i] = b
                new_vectors.append(np.copy(temp))

            poss_vectors = new_vectors

    return poss_vectors

#Проверка на B
"""
Вход:
vector - вектор значений
check_type - тип проверки:
    0:
    Проверяет принадлежит ли классу вектор
    Выход:
        0 - не принадлежит
        1 - принадлежит
        2 - можно дополнить

    1:
    Если вектор принадлежит классу, то выдаёт количество возможных векторов
    Выход:
        0 - не принадлежит
        1 и больше - количество возможных векторов
    2:
    Если вектор принадлежит классу, то выдаёт возможные вектора
    Выход:
        Массив возможных векторов
"""
def B_check(vector, check_type):
    pos_digit = []
    for i in range(len(vector)):
        if vector[i] not in pos_digit:
            pos_digit.append(vector[i])

    res = 0

    if len(vector) == 3:
        res = 1
    elif len(pos_digit) == 4:
        res = 0
    elif len(pos_digit) == 3:
        if 3 in pos_digit:
            res = 2
        else:
            res = 0
    else:
        if 3 in pos_digit:
            res = 2
        else:
            res = 1

    if check_type == 0:
        return res

    elif check_type == 1:
        add_amount = 1
        if len(vector) == 3:
            for val in vector:
                if val == 3:
                    add_amount *= 3
        elif res == 0:
            add_amount = 0
        elif res == 2:
```

```
            for val in vector:
                if val == 3:
                    add_amount *= 2
            if len(pos_digit) == 2:
                add_amount *= 2
                add_amount -= 1
            if len(pos_digit) == 1:
                add_amount -= 1
                add_amount *= 3

        return add_amount

    elif check_type == 2:
        poss_vectors = []

        if len(vector) == 3:
            poss_vectors.append(vector)
            for i in range(len(vector)):
                if vector[i] == 3:
                    new_vectors = []
                    for temp in poss_vectors:
                        temp[i] = 0
                        new_vectors.append(np.copy(temp))

                        temp[i] = 1
                        new_vectors.append(np.copy(temp))

                        temp[i] = 2
                        new_vectors.append(np.copy(temp))

                    poss_vectors = new_vectors
        elif res == 1:
            poss_vectors.append(vector)
        elif res == 2:
            if len(pos_digit) == 1:
                poss_vectors.extend(build_vectors(vector, 0, 1))
                poss_vectors.extend(build_vectors(vector, 0, 2)[1:])
                poss_vectors.extend(build_vectors(vector, 1, 2)[1:-1])
            elif len(pos_digit) == 2:
                a = 0
                for val in pos_digit:
                    if val != 3:
                        a = val
                if a == 0:
                    b, c = 1, 2
                elif a == 1:
                    b, c = 0, 2
                elif a == 2:
                    b, c = 0, 1

                poss_vectors.extend(build_vectors(vector, a, b))
                poss_vectors.extend(build_vectors(vector, a, c)[1:])
            elif len(pos_digit) == 3:
                a, b = -1, -1
                for val in pos_digit:
                    if (val != 3) and (a == -1):
                        a = val
                    elif (val != 3) and (b == -1):
                        b = val
                poss_vectors.extend(build_vectors(vector, a, b))

        poss_vectors = np.array(poss_vectors)
        return poss_vectors




#Построение возможных векторов
"""
```

```
Вход:
vector - вектор значений
"""
def build_poss_vectors(vector, f_digit, s_digit, T_type, isType = False):
    poss_vectors = []
    poss_vectors.append(np.copy(vector))

    if len(vector) == 3:
        for i in range(len(vector)):
            if vector[i] == 3:
                new_vectors = []
                if i == T_type or isType:
                    for temp in poss_vectors:
                        temp[i] = 0
                        new_vectors.append(np.copy(temp))
                        temp[i] = 1
                        new_vectors.append(np.copy(temp))
                        temp[i] = 2
                        new_vectors.append(np.copy(temp))
                    poss_vectors = new_vectors
                else:
                    for temp in poss_vectors:
                        temp[i] = f_digit
                        new_vectors.append(np.copy(temp))

                        temp[i] = s_digit
                        new_vectors.append(np.copy(temp))
                    poss_vectors = new_vectors
    else:
        step = math.ceil(len(vector) / 3)
        k = 0
        for i in range(0, len(vector), step):
            temp = np.copy(vector[i:i + step])
            if 3 in temp:
                if k == T_type:
                    isType = True
                else:
                    isType = False
                temp_vectors  =  build_poss_vectors(temp,  f_digit,  s_digit,  T_type,
isType)
                new_vectors = []
                for vec in poss_vectors:
                    for new_vec in temp_vectors:
                        for j in range(len(new_vec)):
                            vec[i + j] = new_vec[j]
                        new_vectors.append(np.copy(vec))
                poss_vectors = new_vectors
            k += 1
    return poss_vectors



#Проверка на T12, T02, T01
"""
Вход:
vector - вектор значений
check_type - тип проверки:
    0:
    Проверяет принадлежит ли классу вектор
    Выход:
        0 - не принадлежит
        1 - принадлежит
        2 - можно дополнить

    1:
    Если вектор принадлежит классу, то выдаёт количество возможных векторов
    Выход:
        0 - не принадлежит
        1 и больше - количество возможных векторов
```

```
    2:
    Если вектор принадлежит классу, то выдаёт возможные вектора
    Выход:
        Массив возможных векторов
T_type - 0: T12, 1: T02, 2: T01
"""
def T12_T02_T01_check(vector, check_type, T_type):
    f_digit, s_digit = -1, -1
    if T_type == 0:
        f_digit, s_digit = 1, 2
    elif T_type == 1:
        f_digit, s_digit = 0, 2
    elif T_type == 2:
        f_digit, s_digit = 0, 1

    res = 1
    if check_type == 1:
        add_amount = 1
    if len(vector) != 3:
        step = math.ceil(len(vector) / 3)
        j = 0
        res_arr = []
        for i in range(0, len(vector), step):
            temp = np.copy(vector[i:i + step])
            if (j == f_digit) or (j == s_digit):
                res_arr.append(T12_T02_T01_check(temp, 0, T_type))
                if check_type == 1:
                    add_amount *= T12_T02_T01_check(temp, 1, T_type)
            elif check_type == 1:
                for ii in range(len(temp)):
                    if temp[ii] == 3:
                        add_amount *= 3
            j += 1
        if 0 in res_arr:
            res = 0
        elif 2 in res_arr:
            res = 2
    else:
        if (vector[f_digit] == T_type) or (vector[s_digit] == T_type):
            res = 0
            if check_type == 1:
                add_amount *= 0
        elif 3 in vector:
            if (vector[f_digit] == 3) or (vector[s_digit] == 3):
                res = 2
            if check_type == 1:
                for i in range(len(vector)):
                    if (i == T_type) and (vector[i] == 3):
                        add_amount *= 3
                    elif vector[i] == 3:
                        add_amount *= 2

    if check_type == 0:
        return res

    elif check_type == 1:
        return add_amount

    elif check_type == 2:
        poss_vectors = []

        if (res == 1) and (3 not in vector):
            poss_vectors.append(vector)
        elif (res == 1) or (res == 2):
            poss_vectors = build_poss_vectors(vector, f_digit, s_digit, T_type)

        poss_vectors = np.array(poss_vectors)
        return poss_vectors
```

```
#Проверка на S
"""
Вход:
vector - вектор значений
check_type - тип проверки:
    0:
    Проверяет принадлежит ли классу вектор
    Выход:
        0 - не принадлежит
        1 - принадлежит
        2 - можно дополнить

    1:
    Если вектор принадлежит классу, то выдаёт количество возможных векторов
    Выход:
        0 - не принадлежит
        1 и больше - количество возможных векторов
    2:
    Если вектор принадлежит классу, то выдаёт возможные вектора
    Выход:
        Массив возможных векторов
"""
def S_check(vector, check_type):
    len_vec = len(vector)
    step = math.ceil(len_vec / 3)
    n = 0
    while len_vec != 1:
        len_vec = math.ceil(len_vec / 3)
        n += 1

    sets = list(itertools.product(range(3), repeat = n))
    sets = np.array(sets)

    if check_type == 0:
        res_arr = []
        for i in range(step):
            sft1 = (sets[i] + 1) % 3
            sft2 = (sets[i] + 2) % 3

            j1, j2 = 0, 0
            for j in range(len(vector) - step):
                if np.array_equal(sets[step + j], sft1):
                    j1 = step + j
                if np.array_equal(sets[step + j], sft2):
                    j2 = step + j

            res, res1, res2 = 0, 0, 0
            if vector[i] != 3:
                if vector[j1] == (vector[i] + 1) % 3:
                    res1 = 1
                elif vector[j1] == 3:
                    res1 = 2
                if vector[j2] == (vector[i] + 2) % 3:
                    res2 = 1
                elif vector[j2] == 3:
                    res2 = 2
            else:
                if (vector[j2] == (vector[j1] + 1) % 3) or (vector[j2] == 3) or
(vector[j1] == 3):
                    res1 = 2
                    res2 = 2

            if (res1 == 0) or (res2 == 0):
                res = 0
                res_arr.append(res)
                break
            elif (res1 == 2) or (res2 == 2):
```

```
            res = 2
        else:
            res = 1
        res_arr.append(res)

    if 0 in res_arr:
        return 0
    elif 2 in res_arr:
        return 2
    else:
        return 1

elif check_type == 1:
    res = S_check(vector, 0)
    if res != 2:
        return res
    else:
        add_amount = 1
        for i in range(step):
            if vector[i] == 3:
                sft1 = (sets[i] + 1) % 3
                sft2 = (sets[i] + 2) % 3

                j1, j2 = 0, 0
                for j in range(len(vector) - step):
                    if np.array_equal(sets[step + j], sft1):
                        j1 = step + j
                    if np.array_equal(sets[step + j], sft2):
                        j2 = step + j

                if (vector[j1] == 3) and (vector[j2] == 3):
                    add_amount *= 3

        return add_amount

elif check_type == 2:
    poss_vectors = []
    res = S_check(vector, 0)

    if res == 1:
        poss_vectors.append(vector)
    elif res == 2:
        poss_vectors.append(np.copy(vector))
        for i in range(step):
            sft1 = (sets[i] + 1) % 3
            sft2 = (sets[i] + 2) % 3

            j1, j2 = 0, 0
            for j in range(len(vector) - step):
                if np.array_equal(sets[step + j], sft1):
                    j1 = step + j
                if np.array_equal(sets[step + j], sft2):
                    j2 = step + j
            if vector[i] == 3:
                if (vector[j1] == 3) and (vector[j2] == 3):
                    new_vectors = []
                    for temp in poss_vectors:
                        for k in range(3):
                            temp[i] = k
                            temp[j1] = (k + 1) % 3
                            temp[j2] = (k + 2) % 3
                            new_vectors.append(np.copy(temp))
                    poss_vectors = new_vectors
                elif vector[j1] != 3:
                    new_vectors = []
                    for temp in poss_vectors:
                        temp[i] = (vector[j1] + 2) % 3
                        temp[j2] = (vector[j1] + 1) % 3
                        new_vectors.append(np.copy(temp))
                    poss_vectors = new_vectors
```

```
                    else:
                        new_vectors = []
                        for temp in poss_vectors:
                            temp[i] = (vector[j2] + 1) % 3
                            temp[j1] = (vector[j2] + 2) % 3
                            new_vectors.append(np.copy(temp))
                        poss_vectors = new_vectors
                else:
                    new_vectors = []
                    for temp in poss_vectors:
                        temp[j1] = (vector[i] + 1) % 3
                        temp[j2] = (vector[i] + 2) % 3
                        new_vectors.append(np.copy(temp))
                    poss_vectors = new_vectors


        poss_vectors = np.array(poss_vectors)
        return poss_vectors




# Приведение вектора к "нормальному" виду
"""
vector - вектор значений
M_type - 0: M0, 1: M1, 2: M2
"""
def normalize(vector, M_type):
    new_vector = np.zeros(len(vector), dtype = np.int32)

    if len(vector) == 3:
        new_vector[0] = vector[(M_type - 1) % 3]
        new_vector[1] = vector[M_type]
        new_vector[2] = vector[(M_type + 1) % 3]

        for i in range(len(new_vector)):
            if new_vector[i] == M_type:
                new_vector[i] = 5
            elif new_vector[i] == (M_type - 1) % 3:
                new_vector[i] = 4
            elif new_vector[i] == (M_type + 1) % 3:
                new_vector[i] = 6

    else:
        j = 0
        step = math.ceil(len(vector) / 3)
        for i in range(0, len(vector), step):
            temp = np.copy(vector[i:i + step])
            if j == M_type:
                new_vector[step: 2*step] = normalize(temp, M_type)
            elif j == (M_type - 1) % 3:
                new_vector[0:step] = normalize(temp, M_type)
            elif j == (M_type + 1) % 3:
                new_vector[2*step:3*step] = normalize(temp, M_type)
            j += 1

    return new_vector

# Приведение вектора к изначальному виду
"""
vector - нормализованный вектор значений
M_type - 0: M0, 1: M1, 2: M2
"""
def back_normalize(vector, M_type):
    new_vector = np.zeros(len(vector), dtype = np.int32)

    if len(vector) == 3:
        new_vector[(M_type - 1) % 3] = vector[0]
        new_vector[M_type] = vector[1]
```

```
        new_vector[(M_type + 1) % 3] = vector[2]

        for i in range(len(new_vector)):
            if new_vector[i] == 5:
                new_vector[i] = M_type
            elif new_vector[i] == 4:
                new_vector[i] = (M_type - 1) % 3
            elif new_vector[i] == 6:
                new_vector[i] = (M_type + 1) % 3

    else:
        step = math.ceil(len(vector) / 3)
        if M_type == 0:
            new_vector[0:step] = back_normalize(np.copy(vector[step: 2*step]), M_type)
            new_vector[step: 2*step] = back_normalize(np.copy(vector[2*step:3*step]),
M_type)
            new_vector[2*step:3*step] = back_normalize(np.copy(vector[0:step]),
M_type)
        elif M_type == 1:
            new_vector[0:step] = back_normalize(np.copy(vector[0:step]), M_type)
            new_vector[step: 2*step] = back_normalize(np.copy(vector[step: 2*step]),
M_type)
            new_vector[2*step:3*step] = back_normalize(np.copy(vector[2*step:3*step]),
M_type)
        elif M_type == 2:
            new_vector[0:step] = back_normalize(np.copy(vector[2*step:3*step]),
M_type)
            new_vector[step: 2*step] = back_normalize(np.copy(vector[0:step]), M_type)
            new_vector[2*step:3*step] = back_normalize(np.copy(vector[step: 2*step]),
M_type)

    return new_vector

#Нахождение возможных значений
"""
Вход:
low - нижнее значение
high - верхнее значение
"""
def possible_M_values(low, high):
    poss_val = []
    while low <= high:
        poss_val.append(low)
        low += 1
    return poss_val

#Построение возможных векторов M
"""
Вход:
vector - вектор значений
"""
def build_M_vectors(vector):
    poss_vectors = []
    is_error = False
    step = math.ceil(len(vector) / 3)
    temp1 = np.copy(vector[0:step])
    temp2 = np.copy(vector[step:2*step])
    temp3 = np.copy(vector[2*step:3*step])
    new_vectors = []
    new_vectors.append(np.copy(vector))
    for i in range(len(temp1)):
        if ((temp1[i] > temp2[i]) or (temp2[i] > temp3[i])) and temp1[i] != 3 and
temp2[i] != 3 and temp3[i] != 3:
            is_error = True
            break
        elif (temp1[i] <= temp2[i]) and temp3[i] == 3 and temp1[i] != 3 and temp2[i]
!= 3:
            temp_vectors = []
            poss_vals = possible_M_values(temp1[i], 6)
```

```
            for temp in new_vectors:
                for val in poss_vals:
                    temp[i + 2*step] = val
                    temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
        elif (temp1[i] <= temp3[i]) and temp2[i] == 3 and temp1[i] != 3 and temp3[i]
!= 3:
            temp_vectors = []
            poss_vals = possible_M_values(temp1[i], temp3[i])

            for temp in new_vectors:
                for val in poss_vals:
                    temp[i + step] = val
                    temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
        elif (temp2[i] <= temp3[i]) and temp1[i] == 3 and temp2[i] != 3 and temp3[i]
!= 3:
            temp_vectors = []
            poss_vals = possible_M_values(4, temp2[i])

            for temp in new_vectors:
                for val in poss_vals:
                    temp[i] = val
                    temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
        elif temp1[i] == 3 and temp2[i] == 3 and temp3[i] != 3:
            temp_vectors = []
            poss_vals = possible_M_values(4, temp3[i])

            for temp in new_vectors:
                for val in poss_vals:
                    temp[i + step] = val
                    poss_vals1 = possible_M_values(4, val)

                    for val1 in poss_vals1:
                        temp[i] = val1
                        temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
        elif temp1[i] == 3 and temp2[i] != 3 and temp3[i] == 3:
            temp_vectors = []
            poss_vals = possible_M_values(4, temp2[i])
            poss_vals1 = possible_M_values(temp2[i], 6)

            for temp in new_vectors:
                for val in poss_vals:
                    temp[i] = val

                    for val1 in poss_vals1:
                        temp[i + 2*step] = val1
                        temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
        elif temp1[i] != 3 and temp2[i] == 3 and temp3[i] == 3:
            temp_vectors = []
            poss_vals = possible_M_values(temp1[i], 6)

            for temp in new_vectors:
                for val in poss_vals:
                    temp[i + step] = val
                    poss_vals1 = possible_M_values(val, 6)

                    for val1 in poss_vals1:
                        temp[i + 2*step] = val1
                        temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
```

```
        elif temp1[i] == 3 and temp2[i] == 3 and temp3[i] == 3:
            temp_vectors = []
            poss_vals = possible_M_values(4, 6)

            for temp in new_vectors:
                for val in poss_vals:
                    temp[i] = val
                    poss_vals1 = possible_M_values(val, 6)

                    for val1 in poss_vals1:
                        temp[i + step] = val1
                        poss_vals2 = possible_M_values(val1, 6)

                        for val2 in poss_vals2:
                            temp[i + 2*step] = val2
                            temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors

    if not is_error:
        poss_vectors = new_vectors
    return poss_vectors

#Проверка на M0, M1, M2
"""
Вход:
vector - вектор значений
check_type - тип проверки:
    0:
    Проверяет принадлежит ли классу вектор
    Выход:
        0 - не принадлежит
        1 - принадлежит
        2 - можно дополнить

    1:
    Если вектор принадлежит классу, то выдаёт количество возможных векторов
    Выход:
        0 - не принадлежит
        1 и больше - количество возможных векторов
    2:
    Если вектор принадлежит классу, то выдаёт возможные вектора
    Выход:
        Массив возможных векторов
M_type - 0: M0, 1: M1, 2: M2
"""
def M0_M1_M2_check(vector, check_type, M_type, first_in = True):
    norm_vector = np.copy(vector)
    if first_in:
        norm_vector = normalize(vector, M_type)
    if 3 not in norm_vector:
        res = 0

        if len(norm_vector) == 3:
            if   (norm_vector[0]   <=   norm_vector[1])   and   (norm_vector[1]   <=
norm_vector[2]):
                res = 1
        else:
            step = math.ceil(len(norm_vector) / 3)
            temp1 = np.copy(norm_vector[0:step])
            temp2 = np.copy(norm_vector[step:2*step])
            temp3 = np.copy(norm_vector[2*step:3*step])

            arr_res = []
            for i in range(len(temp1)):
                if (temp1[i] <= temp2[i]) and (temp2[i] <= temp3[i]):
                    arr_res.append(1)
                else:
                    arr_res.append(0)
```

```
            if 0 not in arr_res:
                res = 1

            arr_res = []
            if res == 1:
                arr_res.append(M0_M1_M2_check(temp1, 0, M_type, False))
                arr_res.append(M0_M1_M2_check(temp2, 0, M_type, False))
                arr_res.append(M0_M1_M2_check(temp3, 0, M_type, False))
                if 0 in arr_res:
                    res = 0

        if check_type == 0 or check_type == 1:
            return res
        elif check_type == 2:
            poss_vectors = []
            if res == 1 and first_in:
                denorm_vector = back_normalize(norm_vector, M_type)
                poss_vectors.append(denorm_vector)
            elif res == 1:
                poss_vectors.append(norm_vector)
            poss_vectors = np.array(poss_vectors)
            return poss_vectors
    else:
        final_vectors = []
        poss_vectors = build_M_vectors(norm_vector)

        for vec in poss_vectors:
            res = M0_M1_M2_check(vec, 0, M_type, False)
            if res == 1:
                final_vectors.append(back_normalize(vec, M_type))

        if check_type == 0:
            if len(final_vectors) != 0:
                return 2
            else:
                return 0
        elif check_type == 1:
            return len(final_vectors)
        elif check_type == 2:
            final_vectors = np.array(final_vectors)
            return final_vectors




# Приведение вектора к "нормальному" виду
"""
vector - вектор значений
U_type - 0: U0, 1: U1, 2: U2
"""
def U_normalize(vector, U_type):
    new_vector = np.zeros(len(vector), dtype = np.int32)
    f_digit, s_digit = -1, -1
    if U_type == 0:
        f_digit, s_digit = 1, 2
    elif U_type == 1:
        f_digit, s_digit = 0, 2
    elif U_type == 2:
        f_digit, s_digit = 0, 1


    if len(vector) == 3:
        new_vector[0] = vector[U_type]
        new_vector[1] = vector[f_digit]
        new_vector[2] = vector[s_digit]

    else:
        j = 0
        step = math.ceil(len(vector) / 3)
```

```
        for i in range(0, len(vector), step):
            temp = np.copy(vector[i:i + step])
            if j == U_type:
                new_vector[0: step] = U_normalize(temp, U_type)
            elif j == f_digit:
                new_vector[step:2*step] = U_normalize(temp, U_type)
            elif j == s_digit:
                new_vector[2*step:3*step] = U_normalize(temp, U_type)
            j += 1

    return new_vector

# Приведение вектора к изначальному виду
"""
vector - нормализованный вектор значений
U_type - 0: U0, 1: U1, 2: U2
"""
def U_back_normalize(vector, U_type):
    new_vector = np.zeros(len(vector), dtype = np.int32)
    f_digit, s_digit = -1, -1
    if U_type == 0:
        f_digit, s_digit = 1, 2
    elif U_type == 1:
        f_digit, s_digit = 0, 2
    elif U_type == 2:
        f_digit, s_digit = 0, 1

    if len(vector) == 3:
        new_vector[U_type] = vector[0]
        new_vector[f_digit] = vector[1]
        new_vector[s_digit] = vector[2]

    else:
        step = math.ceil(len(vector) / 3)
        if U_type == 0:
            new_vector = np.copy(vector)
        elif U_type == 1:
            new_vector[0:step]     =     U_back_normalize(np.copy(vector[step:2*step]),
U_type)
            new_vector[step:  2*step]   =   U_back_normalize(np.copy(vector[0:  step]),
U_type)
            new_vector[2*step:   3*step]    =    U_back_normalize(np.copy(vector[2*step:
3*step]), U_type)
        elif U_type == 2:
            new_vector[0:step]     =     U_back_normalize(np.copy(vector[step:2*step]),
U_type)
            new_vector[step:                          2*step]                          =
U_back_normalize(np.copy(vector[2*step:3*step]), U_type)
            new_vector[2*step:3*step]   =   U_back_normalize(np.copy(vector[0:  step]),
U_type)

    return new_vector

#Построение возможных векторов U
"""
Вход:
vector - вектор значений
U_type - 0: U0, 1: U1, 2: U2
"""
def build_U_vectors(vector, U_type):
    poss_vectors = []
    is_error = False
    step = math.ceil(len(vector) / 3)
    temp1 = np.copy(vector[0:step])
    temp2 = np.copy(vector[step:2*step])
    temp3 = np.copy(vector[2*step:3*step])
    new_vectors = []
    new_vectors.append(np.copy(vector))
    f_digit, s_digit = -1, -1
    if U_type == 0:
```

```
        f_digit, s_digit = 1, 2
    elif U_type == 1:
        f_digit, s_digit = 0, 2
    elif U_type == 2:
        f_digit, s_digit = 0, 1

    for i in range(len(temp2)):
        if (temp2[i] != temp3[i]) and ((temp2[i] == U_type) or (temp3[i] == U_type))
and temp2[i] != 3 and temp3[i] != 3:
            is_error = True
            break
        elif (temp2[i] == U_type) and temp3[i] == 3:
            temp_vectors = []

            for temp in new_vectors:
                temp[i + 2*step] = U_type
                temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
        elif (temp2[i] != U_type) and temp3[i] == 3 and temp2[i] != 3:
            temp_vectors = []

            for temp in new_vectors:
                temp[i + 2*step] = f_digit
                temp_vectors.append(np.copy(temp))

                temp[i + 2*step] = s_digit
                temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
        elif (temp3[i] == U_type) and temp2[i] == 3:
            temp_vectors = []

            for temp in new_vectors:
                temp[i + step] = U_type
                temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
        elif (temp3[i] != U_type) and temp2[i] == 3 and temp3[i] != 3:
            temp_vectors = []

            for temp in new_vectors:
                temp[i + step] = f_digit
                temp_vectors.append(np.copy(temp))

                temp[i + step] = s_digit
                temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
        elif temp2[i] == 3 and temp3[i] == 3:
            temp_vectors = []

            for temp in new_vectors:
                temp[i + step] = f_digit
                temp[i + 2*step] = f_digit
                temp_vectors.append(np.copy(temp))

                temp[i + step] = s_digit
                temp[i + 2*step] = s_digit
                temp_vectors.append(np.copy(temp))

                temp[i + step] = U_type
                temp[i + 2*step] = U_type
                temp_vectors.append(np.copy(temp))

                temp[i + step] = f_digit
                temp[i + 2*step] = s_digit
                temp_vectors.append(np.copy(temp))

                temp[i + step] = s_digit
```

```
                temp[i + 2*step] = f_digit
                temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors

    if (len(temp1) == 1) and (temp1[0] == 3):
        temp_vectors = []
        for temp in new_vectors:
            temp[0] = 0
            temp_vectors.append(np.copy(temp))

            temp[0] = 1
            temp_vectors.append(np.copy(temp))

            temp[0] = 2
            temp_vectors.append(np.copy(temp))

        new_vectors = temp_vectors

    elif (len(temp1) != 1):
        temp_vectors = []
        short_vectors = build_U_vectors(temp1, U_type)

        for temp in new_vectors:
            for vec in short_vectors:
                temp[0:step] = vec
                temp_vectors.append(np.copy(temp))

        new_vectors = temp_vectors


    if not is_error:
        poss_vectors = new_vectors
    return poss_vectors

#Проверка на U0, U1, U2
"""
Вход:
vector - вектор значений
check_type - тип проверки:
    0:
    Проверяет принадлежит ли классу вектор
    Выход:
        0 - не принадлежит
        1 - принадлежит
        2 - можно дополнить

    1:
    Если вектор принадлежит классу, то выдаёт количество возможных векторов
    Выход:
        0 - не принадлежит
        1 и больше - количество возможных векторов
    2:
    Если вектор принадлежит классу, то выдаёт возможные вектора
    Выход:
        Массив возможных векторов
U_type - 0: U0, 1: U1, 2: U2
"""
def U0_U1_U2_check(vector, check_type, U_type, first_in = True):
    norm_vector = np.copy(vector)
    if first_in:
        norm_vector = U_normalize(vector, U_type)
    if 3 not in norm_vector:
        res = 0

        if len(norm_vector) == 3:
            if (norm_vector[1] == norm_vector[2]) or ((norm_vector[1] != U_type) and
(norm_vector[2] != U_type)):
                res = 1
        else:
```

```
            step = math.ceil(len(norm_vector) / 3)
            temp1 = np.copy(norm_vector[0:step])
            temp2 = np.copy(norm_vector[step:2*step])
            temp3 = np.copy(norm_vector[2*step:3*step])

            arr_res = []
            for i in range(len(temp1)):
                if (temp2[i] == temp3[i]) or ((temp2[i] != U_type) and (temp3[i] !=
U_type)):
                    arr_res.append(1)
                else:
                    arr_res.append(0)

            if 0 not in arr_res:
                res = 1

            arr_res = []
            if res == 1:
                arr_res.append(U0_U1_U2_check(temp1, 0, U_type, False))
                arr_res.append(U0_U1_U2_check(temp2, 0, U_type, False))
                arr_res.append(U0_U1_U2_check(temp3, 0, U_type, False))
                if 0 in arr_res:
                    res = 0

        if check_type == 0 or check_type == 1:
            return res

        elif check_type == 2:
            poss_vectors = []
            if res == 1 and first_in:
                denorm_vector = U_back_normalize(norm_vector, U_type)
                poss_vectors.append(denorm_vector)
            elif res == 1:
                poss_vectors.append(norm_vector)
            poss_vectors = np.array(poss_vectors)
            return poss_vectors
    else:
        final_vectors = []
        poss_vectors = build_U_vectors(norm_vector, U_type)

        for vec in poss_vectors:
            res = U0_U1_U2_check(vec, 0, U_type, False)
            if res == 1:
                final_vectors.append(U_back_normalize(vec, U_type))

        if check_type == 0:
            if len(final_vectors) == 0:
                return 0
            elif (norm_vector[0] == 3) and (3 not in norm_vector[1:]):
                return 1
            else:
                return 2
        elif check_type == 1:
            return len(final_vectors)
        elif check_type == 2:
            final_vectors = np.array(final_vectors)
            return final_vectors




#Построение возможных векторов C
"""
Вход:
vector - вектор значений
C_type - 0: C0, 1: C1, 2: C2
"""
def build_C_vectors(vector, C_type):
    poss_vectors = []
```

```
    is_error = False
    step = math.ceil(len(vector) / 3)
    temp1 = np.copy(vector[0:step])
    temp2 = np.copy(vector[step:2*step])
    temp3 = np.copy(vector[2*step:3*step])
    new_vectors = []
    new_vectors.append(np.copy(vector))
    f_digit, s_digit = -1, -1
    if C_type == 0:
        f_digit, s_digit = 1, 2
    elif C_type == 1:
        f_digit, s_digit = 0, 2
    elif C_type == 2:
        f_digit, s_digit = 0, 1

    for i in range(len(temp1)):
        cond1 = (temp1[i] != temp2[i]) and (temp2[i] != C_type) and (temp1[i] !=
C_type)
        cond2 = (temp1[i] != temp3[i]) and (temp3[i] != C_type) and (temp1[i] !=
C_type)
        cond3 = (temp1[i] != temp2[0]) and (temp2[0] != C_type) and (temp1[i] !=
C_type)
        cond4 = (temp1[i] != temp3[0]) and (temp3[0] != C_type) and (temp1[i] !=
C_type)
        if (cond1 or cond2 or cond3 or cond4) and (temp1[i] != 3) and (temp2[i] != 3)
and (temp3[i] != 3):
            is_error = True
            break
        elif (temp1[i] == 3) and (temp2[i] != 3) and (temp3[i] != 3):
            temp_vectors = []

            if (temp2[i] != temp3[i]) and (temp2[i] != C_type) and (temp3[i] !=
C_type):
                for temp in new_vectors:
                    temp[i] = C_type
                    temp_vectors.append(np.copy(temp))

            elif (temp2[i] != temp3[i]) and ((temp2[i] == C_type) and (temp3[i] ==
C_type)):
                for temp in new_vectors:
                    temp[i] = C_type
                    temp_vectors.append(np.copy(temp))

                    if temp2[i] != C_type:
                        temp[i] = temp2[i]
                        temp_vectors.append(np.copy(temp))
                    else:
                        temp[i] = temp3[i]
                        temp_vectors.append(np.copy(temp))

            elif (temp2[i] == temp3[i]) and (temp2[i] != C_type):
                for temp in new_vectors:
                    temp[i] = C_type
                    temp_vectors.append(np.copy(temp))

                    temp[i] = temp2[i]
                    temp_vectors.append(np.copy(temp))

            elif (temp2[i] == temp3[i]) and (temp2[i] == C_type):
                for temp in new_vectors:
                    temp[i] = C_type
                    temp_vectors.append(np.copy(temp))

                    temp[i] = f_digit
                    temp_vectors.append(np.copy(temp))

                    temp[i] = s_digit
                    temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
```

```
        elif ((temp1[i] != 3) and (temp2[i] == 3) and (temp3[i] != 3)) or ((temp1[i]
!= 3) and (temp2[i] != 3) and (temp3[i] == 3)):
            temp_vectors = []

            plus_val = 0
            if temp2[i] == 3:
                plus_val = step
            else:
                plus_val = 2*step
            if temp1[i] == C_type:
                for temp in new_vectors:
                    temp[i + plus_val] = C_type
                    temp_vectors.append(np.copy(temp))

                    temp[i + plus_val] = f_digit
                    temp_vectors.append(np.copy(temp))

                    temp[i + plus_val] = s_digit
                    temp_vectors.append(np.copy(temp))
            else:
                for temp in new_vectors:
                    temp[i + plus_val] = C_type
                    temp_vectors.append(np.copy(temp))

                    temp[i + plus_val] = temp1[i]
                    temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors
        elif ((temp1[i] == 3) and (temp2[i] == 3) and (temp3[i] != 3)) or ((temp1[i]
== 3) and (temp2[i] != 3) and (temp3[i] == 3)):
            temp_vectors = []

            plus_val = 0
            another_val = 0
            if temp2[i] == 3:
                plus_val = step
                another_val = temp3[i]
            else:
                plus_val = 2*step
                another_val = temp2[i]

            if another_val == C_type:
                for temp in new_vectors:
                    temp[i] = C_type
                    temp[i + plus_val] = C_type
                    temp_vectors.append(np.copy(temp))

                    temp[i] = C_type
                    temp[i + plus_val] = f_digit
                    temp_vectors.append(np.copy(temp))

                    temp[i] = C_type
                    temp[i + plus_val] = s_digit
                    temp_vectors.append(np.copy(temp))

                    temp[i] = f_digit
                    temp[i + plus_val] = C_type
                    temp_vectors.append(np.copy(temp))

                    temp[i] = f_digit
                    temp[i + plus_val] = f_digit
                    temp_vectors.append(np.copy(temp))

                    temp[i] = s_digit
                    temp[i + plus_val] = C_type
                    temp_vectors.append(np.copy(temp))

                    temp[i] = s_digit
                    temp[i + plus_val] = s_digit
                    temp_vectors.append(np.copy(temp))
```

```
    else:
        for temp in new_vectors:
            temp[i] = C_type
            temp[i + plus_val] = C_type
            temp_vectors.append(np.copy(temp))

            temp[i] = C_type
            temp[i + plus_val] = f_digit
            temp_vectors.append(np.copy(temp))

            temp[i] = C_type
            temp[i + plus_val] = s_digit
            temp_vectors.append(np.copy(temp))

            temp[i] = another_val
            temp[i + plus_val] = C_type
            temp_vectors.append(np.copy(temp))

            temp[i] = another_val
            temp[i + plus_val] = another_val
            temp_vectors.append(np.copy(temp))

    new_vectors = temp_vectors
elif (temp1[i] != 3) and (temp2[i] == 3) and (temp3[i] == 3):
    temp_vectors = []

    if temp1[i] == C_type:
        poss_vals = [C_type, f_digit, s_digit]
        for temp in new_vectors:
            for val1 in poss_vals:
                for val2 in poss_vals:
                    temp[i + step] = val1
                    temp[i + 2*step] = val2
                    temp_vectors.append(np.copy(temp))

    else:
        poss_vals = [C_type, temp1[i]]
        for temp in new_vectors:
            for val1 in poss_vals:
                for val2 in poss_vals:
                    temp[i + step] = val1
                    temp[i + 2*step] = val2
                    temp_vectors.append(np.copy(temp))

    new_vectors = temp_vectors
elif (temp1[i] == 3) and (temp2[i] == 3) and (temp3[i] == 3):
    temp_vectors = []

    for temp in new_vectors:
        temp[i] = C_type
        poss_vals = [C_type, f_digit, s_digit]
        for val1 in poss_vals:
            for val2 in poss_vals:
                temp[i + step] = val1
                temp[i + 2*step] = val2
                temp_vectors.append(np.copy(temp))

        temp[i] = f_digit
        poss_vals = [C_type, f_digit]
        for val1 in poss_vals:
            for val2 in poss_vals:
                temp[i + step] = val1
                temp[i + 2*step] = val2
                temp_vectors.append(np.copy(temp))

        temp[i] = s_digit
        poss_vals = [C_type, s_digit]
        for val1 in poss_vals:
            for val2 in poss_vals:
                temp[i + step] = val1
```

```
                        temp[i + 2*step] = val2
                        temp_vectors.append(np.copy(temp))

            new_vectors = temp_vectors

    if not is_error:
        poss_vectors = new_vectors
    return poss_vectors

#Проверка на C0, C1, C2
"""
Вход:
vector - вектор значений
check_type - тип проверки:
    0:
    Проверяет принадлежит ли классу вектор
    Выход:
        0 - не принадлежит
        1 - принадлежит
        2 - можно дополнить

    1:
    Если вектор принадлежит классу, то выдаёт количество возможных векторов
    Выход:
        0 - не принадлежит
        1 и больше - количество возможных векторов
    2:
    Если вектор принадлежит классу, то выдаёт возможные вектора
    Выход:
        Массив возможных векторов
C_type - 0: C0, 1: C1, 2: C2
"""
def C0_C1_C2_check(vector, check_type, C_type, first_in = True):
    norm_vector = np.copy(vector)
    if first_in:
        norm_vector = U_normalize(vector, C_type)
    if 3 not in norm_vector:
        res = 0

        if len(norm_vector) == 3:
            cond1 = (norm_vector[0] == norm_vector[1]) or (norm_vector[1] == C_type)
            cond2 = (norm_vector[0] == norm_vector[2]) or (norm_vector[2] == C_type)
            if norm_vector[0] == C_type:
                res = 1
            elif cond1 and cond2:
                res = 1
        else:
            step = math.ceil(len(norm_vector) / 3)
            temp1 = np.copy(norm_vector[0:step])
            temp2 = np.copy(norm_vector[step:2*step])
            temp3 = np.copy(norm_vector[2*step:3*step])

            arr_res = []
            for i in range(len(temp2)):
                cond1 = (temp1[i] == temp2[i]) or (temp2[i] == C_type) or (temp1[i] ==
C_type)
                cond2 = (temp1[i] == temp3[i]) or (temp3[i] == C_type) or (temp1[i] ==
C_type)
                cond3 = (temp1[i] == temp2[0]) or (temp2[0] == C_type) or (temp1[i] ==
C_type)
                cond4 = (temp1[i] == temp3[0]) or (temp3[0] == C_type) or (temp1[i] ==
C_type)
                if cond1 and cond2 and cond3 and cond4:
                    arr_res.append(1)
                else:
                    arr_res.append(0)

            if 0 not in arr_res:
                res = 1
```

```
            arr_res = []
            if res == 1:
                arr_res.append(C0_C1_C2_check(temp1, 0, C_type, False))
                arr_res.append(C0_C1_C2_check(temp2, 0, C_type, False))
                arr_res.append(C0_C1_C2_check(temp3, 0, C_type, False))
                if 0 in arr_res:
                    res = 0

        if check_type == 0 or check_type == 1:
            return res

        elif check_type == 2:
            poss_vectors = []
            if res == 1 and first_in:
                denorm_vector = U_back_normalize(norm_vector, C_type)
                poss_vectors.append(denorm_vector)
            elif res == 1:
                poss_vectors.append(norm_vector)
            poss_vectors = np.array(poss_vectors)
            return poss_vectors
    else:
        final_vectors = []
        poss_vectors = build_C_vectors(norm_vector, C_type)

        for vec in poss_vectors:
            res = C0_C1_C2_check(vec, 0, C_type, False)
            if res == 1:
                final_vectors.append(U_back_normalize(vec, C_type))

        if check_type == 0:
            if len(final_vectors) == 0:
                return 0
            else:
                return 2
        elif check_type == 1:
            return len(final_vectors)
        elif check_type == 2:
            final_vectors = np.array(final_vectors)
            return final_vectors




#Проверка на L
"""
Вход:
vector - вектор значений
check_type - тип проверки:
    0:
    Проверяет принадлежит ли классу вектор
    Выход:
        0 - не принадлежит
        1 - принадлежит
        2 - можно дополнить

    1:
    Если вектор принадлежит классу, то выдаёт количество возможных векторов
    Выход:
        0 - не принадлежит
        1 и больше - количество возможных векторов
    2:
    Если вектор принадлежит классу, то выдаёт возможные вектора
    Выход:
        Массив возможных векторов
"""
def L_check(vector, check_type):
    len_vec = len(vector)
    n = 0
    while len_vec >= 3:
```

```
    len_vec = math.ceil(len_vec / 3)
    n += 1
sets = np.array(list(itertools.product(range(3), repeat = n)))
checked_list = np.zeros(len(vector), dtype = np.int32)
if 3 not in vector:
    a0 = vector[0]
    x_list = []
    checked_list[0] = 1
    res = 0
    for i in range(len(vector[1:])):
        if sum(sets[i + 1]) == 1:
            x_list.append(i+1)
            checked_list[i+1] = 1
    y_list = []
    res_arr = []
    for ind in x_list:
        if (vector[2*ind] == ((2 * (vector[ind] - a0) + a0) % 3)):
            res_arr.append(1)
            y_list.append(2*ind)
            checked_list[2*ind] = 1
        else:
            res_arr.append(0)
            checked_list[2*ind] = -1

    if 0 not in res_arr:
        i = 0
        flag = False
        for s in sets:
            if checked_list[i] == 0:
                point_sum = a0
                j = len(x_list) - 1
                for digit in s:
                    if digit == 1:
                        point_sum += vector[x_list[j]] - a0
                    elif digit == 2:
                        point_sum += vector[y_list[j]] - a0
                    j -= 1
                point_sum = point_sum % 3
                if vector[i] == point_sum:
                    res_arr.append(1)
                    checked_list[i] = 1
                else:
                    flag = True
                    res_arr.append(0)
                    checked_list[i] = -1
            if flag:
                break
            i += 1

    if 0 not in res_arr:
        res = 1
    if (check_type == 0) or (check_type == 1):
        return res
    else:
        poss_vectors = []
        if res == 1:
            poss_vectors.append(vector)
        poss_vectors = np.array(poss_vectors)
        return poss_vectors
else:
    poss_vectors = []
    poss_vectors.append(np.copy(vector))
    x_list = []
    is_error = False
    for i in range(len(vector)):
        if sum(sets[i]) == 1:
            x_list.append(i)
    x_list = np.array(x_list)
    y_list = x_list * 2
    if vector[0] == 3:
```

```
        k = 0
        for i in x_list:
            if vector[i] == 3 and vector[y_list[k]] == 3:
                temp_vectors = []
                poss_vals = [0, 1, 2]
                for temp in poss_vectors:
                    for val1 in poss_vals:
                        for val2 in poss_vals:
                            temp[0] = val1
                            temp[i] = val2
                            temp[y_list[k]] = (2 * val2 - val1) % 3
                            temp_vectors.append(np.copy(temp))

                poss_vectors = temp_vectors
            elif vector[i] == 3 and vector[y_list[k]] != 3:
                temp_vectors = []
                poss_vals = [0, 1, 2]
                for val in poss_vals:
                    if vector[y_list[k]] == val:
                        for temp in poss_vectors:
                            temp[0] = val
                            temp[i] = val
                            temp_vectors.append(np.copy(temp))

                            temp[0] = (val + 1) % 3
                            temp[i] = (val + 2) % 3
                            temp_vectors.append(np.copy(temp))

                            temp[0] = (val + 2) % 3
                            temp[i] = (val + 1) % 3
                            temp_vectors.append(np.copy(temp))
                poss_vectors = temp_vectors
            elif vector[i] != 3 and vector[y_list[k]] == 3:
                temp_vectors = []
                poss_vals = [0, 1, 2]
                for temp in poss_vectors:
                    for val in poss_vals:
                        temp[0] = val
                        temp[y_list[k]] = (2 * vector[i] - val) % 3
                        temp_vectors.append(np.copy(temp))
                poss_vectors = temp_vectors
            elif vector[i] != 3 and vector[y_list[k]] != 3:
                temp_vectors = []
                for temp in poss_vectors:
                    temp[0] = (2 * vector[i] - vector[y_list[k]]) % 3
                    temp_vectors.append(np.copy(temp))
                poss_vectors = temp_vectors
            k += 1
    else:
        k = 0
        for i in x_list:
            if vector[i] == 3 and vector[y_list[k]] == 3:
                temp_vectors = []
                poss_vals = [0, 1, 2]
                for temp in poss_vectors:
                    for val in poss_vals:
                        temp[i] = val
                        temp[y_list[k]] = (2 * val - vector[0]) % 3
                        temp_vectors.append(np.copy(temp))
                poss_vectors = temp_vectors
            elif vector[i] == 3 and vector[y_list[k]] != 3:
                temp_vectors = []
                for temp in poss_vectors:
                    temp[i] = (2 * (vector[0] + vector[y_list[k]])) % 3
                    temp_vectors.append(np.copy(temp))
                poss_vectors = temp_vectors
            elif vector[i] != 3 and vector[y_list[k]] == 3:
                temp_vectors = []
                for temp in poss_vectors:
                    temp[y_list[k]] = (2 * vector[y_list[k]] - vector[0]) % 3
```

```
                        temp_vectors.append(np.copy(temp))
                    poss_vectors = temp_vectors
                k += 1

        i = 0
        for s in sets:
            if(poss_vectors[0][i] == 3):
                for temp in poss_vectors:
                    point_sum = temp[0]
                    j = len(x_list) - 1
                    for digit in s:
                        if digit == 1:
                            point_sum += temp[x_list[j]] - temp[0]
                        elif digit == 2:
                            point_sum += temp[y_list[j]] - temp[0]
                        j -= 1
                    point_sum = point_sum % 3
                    temp[i] = point_sum
            i += 1

        final_vectors = []

        for vec in poss_vectors:
            res = L_check(vec, 0)
            if res == 1:
                final_vectors.append(vec)

        if check_type == 0:
            if len(final_vectors) == 0:
                return 0
            else:
                return 2
        elif check_type == 1:
            return len(final_vectors)
        elif check_type == 2:
            final_vectors = np.array(final_vectors)
            return final_vectors




def test_for_class(test_class, vector, check_type):
    a = 0
    if test_class == 'T0':
        a = T0_T1_T2_check(vector, check_type, 0)
    elif test_class == 'T1':
        a = T0_T1_T2_check(vector, check_type, 1)
    elif test_class == 'T2':
        a = T0_T1_T2_check(vector, check_type, 2)
    elif test_class == 'T12':
        a = T12_T02_T01_check(vector, check_type, 0)
    elif test_class == 'T02':
        a = T12_T02_T01_check(vector, check_type, 1)
    elif test_class == 'T01':
        a = T12_T02_T01_check(vector, check_type, 2)
    elif test_class == 'B':
        a = B_check(vector, check_type)
    elif test_class == 'S':
        a = S_check(vector, check_type)
    elif test_class == 'L':
        a = L_check(vector, check_type)
    elif test_class == 'M0':
        a = M0_M1_M2_check(vector, check_type, 0)
    elif test_class == 'M1':
        a = M0_M1_M2_check(vector, check_type, 1)
    elif test_class == 'M2':
        a = M0_M1_M2_check(vector, check_type, 2)
    elif test_class == 'U0':
        a = U0_U1_U2_check(vector, check_type, 0)
```

```
    elif test_class == 'U1':
        a = U0_U1_U2_check(vector, check_type, 1)
    elif test_class == 'U2':
        a = U0_U1_U2_check(vector, check_type, 2)
    elif test_class == 'C0':
        a = C0_C1_C2_check(vector, check_type, 0)
    elif test_class == 'C1':
        a = C0_C1_C2_check(vector, check_type, 1)
    elif test_class == 'C2':
        a = C0_C1_C2_check(vector, check_type, 2)
    return a




def test_for_classes(input_path, check_type, union_type):
    vector, classes = read_input(input_path)
    is_input_correct = True
    if (len(vector) % 3 != 0):
        print("Неверная длина вектора!")
        is_input_correct = False
    else:
        for val in vector:
            if (val != 0) and (val != 1) and (val != 2) and (val != 3):
                print("Неверные значения вектора!")
                is_input_correct = False

    if (check_type != 0) and (check_type != 1) and (check_type != 2):
        print("Неверный тип проверки!")
        is_input_correct = False

    if (union_type != 0) and (union_type != 1):
        print("Неверный тип объединения!")
        is_input_correct = False

    check_res = []
    for cl in classes:
        cond1 = (cl == 'T0') or (cl == 'T1') or (cl == 'T2') or (cl == 'T12') or (cl
== 'T02') or (cl == 'T01')
        cond2 = (cl == 'C0') or (cl == 'C1') or (cl == 'C2') or (cl == 'U0') or (cl ==
'U1') or (cl == 'U2')
        cond3 = (cl == 'M0') or (cl == 'M1') or (cl == 'M2') or (cl == 'S') or (cl ==
'B') or (cl == 'L')
        if cond1 or cond2 or cond3:
            check_res.append(1)
        else:
            check_res.append(0)
    if 0 in check_res:
        print("Одно из названий класса неверное")
        is_input_correct = False

    if is_input_correct:
        result = 0
        unresult = 0
        if len(classes) == 1:
            result = test_for_class(classes[0], vector, check_type)
        else:
            if check_type == 0:
                res_arr = []
                for cl in classes:
                    res_arr.append(test_for_class(cl, vector, 0))
                if union_type == 0:
                    result = np.copy(res_arr)
                else:
                    if (0 not in res_arr) and (2 not in res_arr):
                        unresult = 1
                    elif (2 in res_arr) and (0 not in res_arr):
                        i = 0
                        final_vectors = []
```

```
                        for cl in classes:
                            temp_vectors = []
                            if i == 0:
                                temp_vectors = test_for_class(cl, vector, 2)
                            else:
                                for temp in final_vectors:
                                    if (test_for_class(cl, temp, 0) == 1):
                                        temp_vectors.append(temp)
                            final_vectors = temp_vectors.copy()
                            i += 1
                        if len(final_vectors) != 0:
                            unresult = 2
            elif check_type == 1:
                if union_type == 0:
                    res_arr = []
                    for cl in classes:
                        res_arr.append(test_for_class(cl, vector, 1))
                    result = np.copy(res_arr)
                else:
                    i = 0
                    final_vectors = []
                    for cl in classes:
                        temp_vectors = []
                        if i == 0:
                            temp_vectors = test_for_class(cl, vector, 2)
                        else:
                            for temp in final_vectors:
                                if (test_for_class(cl, temp, 0) == 1):
                                    temp_vectors.append(temp)
                        final_vectors = temp_vectors.copy()
                        i += 1
                    unresult = len(final_vectors)

            elif check_type == 2:
                i = 0
                final_vectors = []
                for cl in classes:
                    temp_vectors = []
                    if i == 0:
                        temp_vectors = test_for_class(cl, vector, 2)
                    else:
                        for temp in final_vectors:
                            if (test_for_class(cl, temp, 0) == 1):
                                temp_vectors.append(temp)
                    final_vectors = temp_vectors.copy()
                    i += 1
                unresult = np.array(final_vectors)
        if (len(classes) == 1):
            if check_type == 0:
                if result == 1:
                    print(f"Функция {vector} лежит в классе {classes[0]}")
                elif result == 2:
                    print(f"Функцию {vector} можно дополнить до принадлежности классу
{classes[0]}")
                elif result == 0:
                    print(f"Функция {vector} не лежит в классе {classes[0]}")
            elif check_type == 1:
                if result == 0:
                    print(f"Функция {vector} не лежит в классе {classes[0]}")
                else:
                    print(f"Из вектора значений {vector} можно получить {result}
функций, лежащих в классе {classes[0]}")
            elif check_type == 2:
                if len(result) == 0:
                    print(f"Функция {vector} не лежит в классе {classes[0]}")
                else:
                    print(f"Из вектора значений {vector} можно получить следующие
функции, лежащие в классе {classes[0]}:")
                    for vec in result:
                        print(vec)
```

```
        elif (union_type == 0):
            if check_type == 0:
                print("Принадлежность классу: ")
                for i in range(len(classes)):
                    if result[i] == 1:
                        print(f"Функция {vector} лежит в классе {classes[i]}")
                    elif result[i] == 2:
                        print(f"Функцию {vector} можно дополнить до принадлежности
классу {classes[i]}")
                    elif result[i] == 0:
                        print(f"Функция {vector} не лежит в классе {classes[i]}")
            elif check_type == 1:
                print("Количество возможных функций: ")
                for i in range(len(classes)):
                    if result[i] == 0:
                        print(f"Функция {vector} не лежит в классе {classes[i]}")
                    else:
                        print(f"Из вектора значений {vector} можно получить
{result[i]} функций, лежащих в классе {classes[i]}")
            elif check_type == 2:
                if len(unresult) == 0:
                    print(f"Функция {vector} не лежит в объединение классов
{classes}")
                else:
                    print(f"Из вектора значений {vector} можно получить следующие
функции, лежащие в объединение классов {classes}:")
                    for vec in unresult:
                        print(vec)
        else:
            if check_type == 0:
                if unresult == 1:
                    print(f"Функция {vector} лежит в объединение классов {classes}")
                elif unresult == 2:
                    print(f"Функцию {vector} можно дополнить до принадлежности
объединению классов {classes}")
                elif unresult == 0:
                    print(f"Функция {vector} не лежит в объединение классов
{classes}")
            elif check_type == 1:
                if unresult == 0:
                    print(f"Функция {vector} не лежит в объединение классов
{classes}")
                else:
                    print(f"Из вектора значений {vector} можно получить {unresult}
функций, лежащих в объединение классов {classes}")
            elif check_type == 2:
                if len(unresult) == 0:
                    print(f"Функция {vector} не лежит в объединение классов
{classes}")
                else:
                    print(f"Из вектора значений {vector} можно получить следующие
функции, лежащие в объединение классов {classes}:")
                    for vec in unresult:
                        print(vec)



def main():
    file_name = str(sys.argv[1])
    check_type = int(sys.argv[2])
    union_type = 0
    if len(sys.argv) == 4:
        union_type = int(sys.argv[3])
    test_for_classes(file_name, check_type, union_type)

if __name__ == '__main__':
    main()
```