МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

федеральное государственное автономное образовательное учреждение высшего образования

«Санкт–Петербургский государственный университет  
аэрокосмического приборостроения»

ФАКУЛЬТЕТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ОТЧЕТ О ПРАКТИКЕ   
ЗАЩИЩЕН С ОЦЕНКОЙ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

РУКОВОДИТЕЛЬ И.Г. Бартасевич

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| преподаватель |  |  |  | У.С. Опалева |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ |
| В СОСТАВЕ ПРОФЕССИОНАЛЬНОГО МОДУЛЯ  ПМ 02 Осуществление интеграции программных модулей\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *код и наименование профессионального модуля* |
|  |
|  |
|  |

ОТЧЕТ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | С022 |  | 24.05.2023 |  | Я.Р. Калашников |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на прохождение учебной практики обучающегося по специальности

09.02.07 Информационные системы и программирование

*код и наименование специальности*

1. Фамилия, имя, отчество обучающегося:

Калашников Ян Русланович

1. Группа: С022 Сроки проведения практики: с «11» мая 2023г. по «24» мая 2023г.
2. Тема задания: приобретение первичных профессиональных умений и навыков, начального опыта практической деятельности, овладение необходимыми компетенциями по профессиональному модулю:

ПМ 02 Осуществление интеграции программных модулей

1. Вопросы, подлежащие изучению:

* разработка структуры проекта, построение диаграмм UML;
* решение задачи математического моделирования;
* подключение системы контроля версий Git;
* выполнение интеграции программных модулей;
* написание модульных тестов.

1. Выполнение комплексных работ по созданию проекта на основе фреймворка Bottle, выполнению интеграции шаблонов, обработчиков форм и файлов данных, использованию методов для получения кода на языке Python с заданной функциональностью и степенью качества, автоматизации тестирования посредством платформы unittest.
2. Содержание отчетной документации:
   1. Отчёт, включающий в себя:

* титульный лист;
* индивидуальное задание;
* материалы о выполнении индивидуального задания;
* список использованных источников.
  1. Аттестационный лист.

1. Срок представления отчета заместителю декана по учебно–производственной работе: «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_\_ г.

Руководитель практики от факультета СПО:

У.С. Опалева

преподаватель 11.05.2023 И. Г. Бартасевич

должность, ученая степень, звание подпись, дата инициалы, фамилия

Обучающийся

11.05.2023 Я.Р. Калашников\_\_\_\_\_

дата подпись инициалы, фамилия

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc134706171)

[1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 5](#_Toc134706172)

[1.1 Введение 5](#_Toc134706173)

[1.2 Основаниядляразработки 5](#_Toc134706174)

[1.3 Назначениеразработки 5](#_Toc134706175)

[1.4 Требования к программе или программному изделию 5](#_Toc134706176)

[1.4.1 Требования к графическому дизайну сайта 5](#_Toc134706177)

[1.4.2 Требования к функциональным характеристикам 5](#_Toc134706178)

[1.5 Требования к эргономике и технической эстетике 5](#_Toc134706179)

[1.6 Стадии и этапы разработки 5](#_Toc134706180)

# **ВВЕДЕНИЕ**

Решение широкого круга задач требует умения разрабатывать структуру проекта, строить диаграммы UML, работать с системами контроля версий, интегрировать программные модули, писать модульные тесты, решать задачи математического моделирования.

Задачей учебной практики по осуществлению интеграции программных продуктов по МДК 02 является закрепление теоретического материала междисциплинарного курса и получение практических навыков по интегрированию программных модулей и решению задач математического моделирования.

# 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

# 1.1 Введение

Настоящее техническое задание распространяется на разработку сайта, предназначенного для решения математических задач с элементами теории графов, неориентированными графами, эйлеровыми графами.

# 1.2 Основания для разработки

Программа разрабатывается на основе учебной практики по ПМ02 «Осуществление интеграции программных модулей».

# 1.3 Назначение разработки

Основным назначением сайта является решение математических задач с элементами теории графов, неориентированными графами и Эйлеровыми графами.

# 1.4 Требования к программе или программному изделию

# 1.4.1 Требования к графическому дизайну сайта

Доступ к основным разделам сайта должен быть доступен на главной странице.

С каждой страницы есть возможность перехода на остальные разделы сайта.

На каждой странице с решением задачи располагается краткая информация по задаче, способу ее решения (ссылка на внешний ресурс).

Оформление должно быть минималистичным, без ярких броских цветов, отвечать современным стандартам дизайна.

# 1.4.2 Требования к функциональным характеристикам

# 1.4.2.1 Классы пользователей

Основной пользователь – гость. Он обладает следующими возможностями:

* переход между разделами сайта;
* просмотр и редактирование данных на странице с поиском подграфа, просмотр результата;
* просмотр и редактирование данных на странице с поиском вершин, имеющих наибольшее окружение;
* просмотр и редактирование данных на странице с поиском Эйлерова цикла в Эйлеровом графе.

# 1.4.2.2 Требования к функциональной части

Требуется найти заданный подграф в данном графе.

Созданная программа должна выполнять следующие действия:

* Поиск всех имеющихся заданных подграфов в исходном графе;
* Вывод всех найденных подграфов в виде списка вершин.

# 1.4.2.3 Исходные данные

Количество вершин графа, матрица смежности вершин.

# 1.4.2.3 Выходные данные

Файл с введенными данными пользователя, результат работы алгоритма, а также время выполнения запроса.

# 1.4.2.4 Требования к представлению сайта

Сайт состоит из главной страницы с описанием рабочей области, а также кнопками для перехода на разделы сайта с решением задач.

На каждой странице с решением задачи находится краткая информация по методу решения, матрица смежности графа, кнопки, позволяющие заполнить таблицу случайными значениями и получить результат.

На странице с авторами находятся фотографии каждого участника команды с коротким описанием личной информации и вкладом, внесенным в разработку сайта.

На каждой странице есть возможность перемещаться между разделами сайта с помощью панели навигации сверху сайта.

# 1.4.3 Требования к техническому обеспечению

Скорость интернет-соединения не менее 500кб/с;

Компьютер с выходом в интернет и одним из популярных браузеров (Chrome, Edge, Yandex, Mozilla, Safari). Операционная система не ниже Windows 7. Оперативная память не ниже 4 гб. Процессор не ниже Intel Core 2 Duo. Встроенный или дискретный графический процессор.

# 1.4.4 Требования к надежности

Все действия пользователя должны быть продуманы, сайт не должен прекращать работу даже в случае некорректных действий. Должны выдаваться сообщения об ошибках при вводе некорректных значений.

# 1.5 Требования к эргономике и технической эстетике

Элементы сайта должны быть оптимизированы под популярные разрешения экранов без искажения основных структурных элементов. Тексты должны быть хорошо различимы и читаемы. Кнопки, метки и остальные элементы должны отвечать требованиям современного дизайна. Картинки не должны быть излишне яркими и броскими. На каждой странице есть название сайта. С каждой страницы доступен переход по основным разделам.

# 2. Алгоритм задачи

Листинг алгоритмов:

def find\_cliques(graph, size):

"""Функция для поиска уникальных клик заданного размера в графе."""

cliques = set()

nodes = set(graph)

for start\_node in nodes:

subgraph = nodes.intersection(graph[start\_node])

for clique in find\_subcliques(graph, [start\_node], subgraph, size):

cliques.add(tuple(sorted(clique)))

cliques = [list(clique) for clique in cliques]

num\_cliques = len(cliques)

return num\_cliques, cliques

def find\_subcliques(graph, prev\_nodes, nodes, size):

"""Функция для поиска подклик заданного размера."""

if len(prev\_nodes) == size:

yield prev\_nodes

else:

for node in nodes:

if all(node in graph[prev\_node] for prev\_node in prev\_nodes):

for clique in find\_subcliques(graph, prev\_nodes + [node], nodes.intersection(graph[node]), size):

yield clique

Блок-схема:



Рисунок 1 – Алгоритм метода find\_cliques



Рисунок 2 – Алгоритм метода find\_subcliques

Данный код является реализацией алгоритма поиска уникальных клик заданного размера в графе.

Граф представляется в виде словаря, где ключами являются вершины графа, а значениями являются множества вершин, которые соединены ребром с данным ключом.

Функция find\_cliques принимает на вход граф и размер клик, которые нужно найти. В цикле она перебирает все вершины графа и для каждой из них находит множество смежных вершин. Затем для каждой найденной клики вызывается функция find\_subcliques, которая рекурсивно ищет все подклики заданного размера. Найденные клики сохраняются в множество cliques, после чего множество преобразуется в список и возвращается вместе с количеством найденных клик.

Функция find\_subcliques также принимает на вход граф, список вершин предыдущей клики, множество вершин для текущей клики и размер клики. Если список предыдущих вершин равен заданному размеру, то возвращается сам список вершин. Иначе перебираются все вершины из множества вершин для текущей клики, которые соединены с каждой вершиной предыдущей клики. Для каждой найденной вершины вызывается рекурсивно функция find\_subcliques с новым списком вершин для текущей клики и новым множеством вершин, которые соединены с найденной вершиной. Результаты всех найденных клик возвращаются в виде сохраненного состояния функции.

# 3. Разработка UML-диаграммы Use Case и компонентов приложения

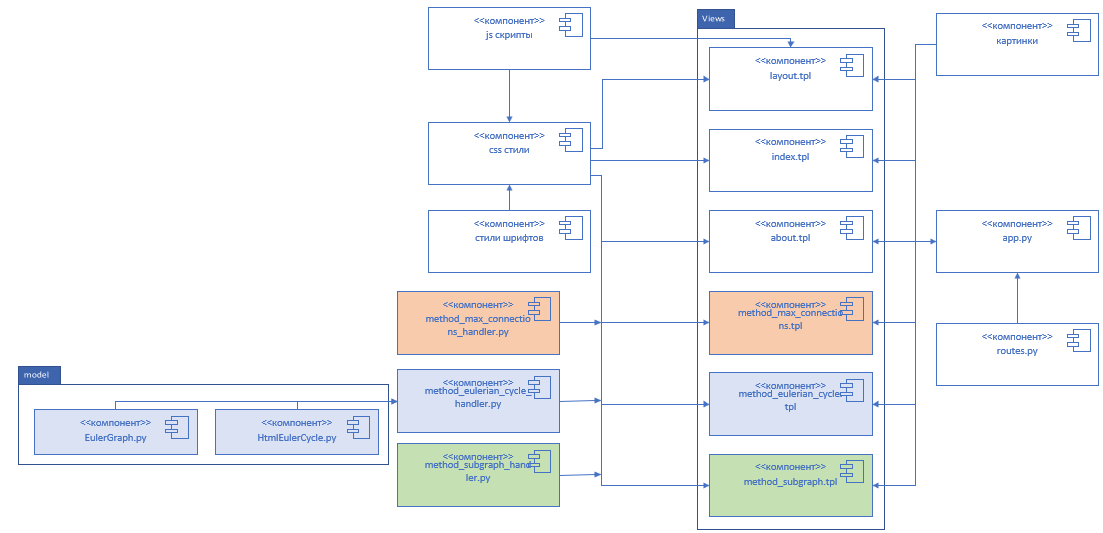


Рисунок 3 – Диаграмма компонентов

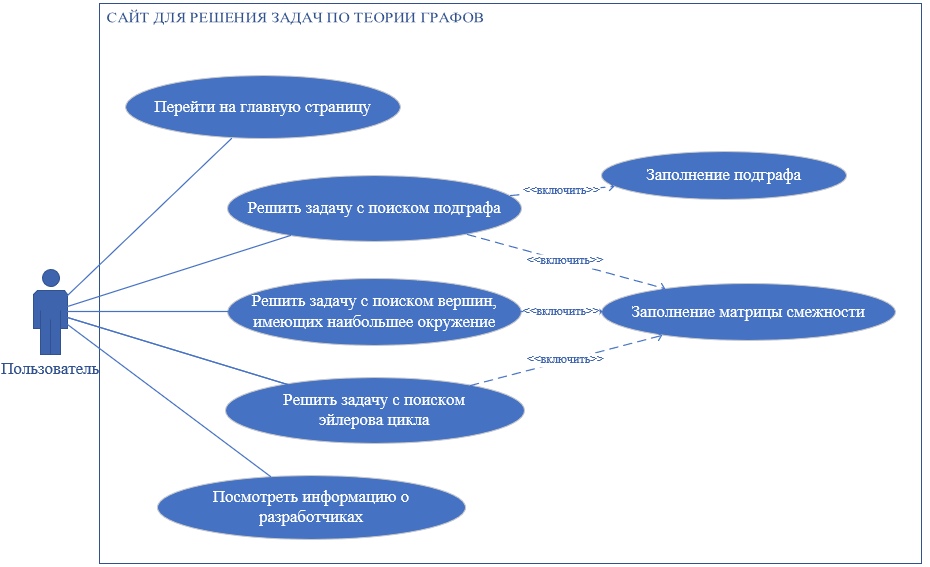


Рисунок 4 – Диаграмма Use Case

# 4. Структура проекта

# 5. Разработка и тестирование страницы с решением задачи

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

# ПРИЛОЖЕНИЕ А. Структура файла README

# BottleWebProject\_C022\_ 2\_KRT

## Реализованная функциональность

- [X] \*\*Разработан метод для поиска подграфа в заданном графе\*\*

- [X] \*\*Разработан метод для поиска вершин с наибольшим окружением\*\*

- [X] \*\*Разработан метод для поиска эйлерова цикла\*\*

- [X] \*\*Разработана UML диаграмма компонентов\*\*

- [X] \*\*Разработана USE CASE диаграмма\*\*

- [X] \*\*Оформлена стартовая страница\*\*

- [X] \*\*Оформлена страница с информацией о разработчиках\*\*

- [X] \*\*Оформлен дизайн страниц с решением методов\*\*

## Используемые технологии

1. Веб-технологии

- HTML

- CSS

- Фреймворк Bottle

- Bootstrap

2. Языки программирования

- Python

- Java

3. Удаленный репозиторий

- GitHub

## Среда разработки:

- Visual Studio

- PyCharm

## Способы установки проекта

1. Клонировать проект с помощью ссылки

2. Открытие проекта в GitHub Desktop

3. Скачивание zip-архива

## Разработчики

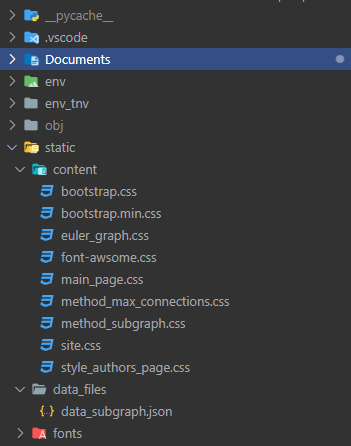
[Hatemsla](https://github.com/Hatemsla "GitHub профиль")

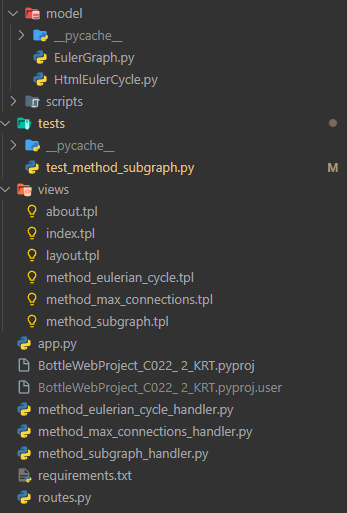
[kirushaaxd](https://github.com/kirushaaxd "GitHub профиль")

[TNV-801](https://github.com/TNV-801 "GitHub профиль")

# ПРИЛОЖЕНИЕ Б. Структура проекта и отслеживание Git-репозитория

Структура проекта:





# ПРИЛОЖЕНИЕ В. Листинг программного кода приложения

# ПРИЛОЖЕНИЕ Д. Листинги юнит-тестов

import method\_subgraph\_handler

import unittest

class TestPositiveFindCliquesMethods(unittest.TestCase):

# Positive

def test\_find\_cliques\_\_3\_size\_clique\_and\_5\_size\_graph\_\_list\_of\_1\_2\_3\_in\_found\_cliques(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2],

4: [5],

5: [4]

}

size = 3

expected\_cliques = [1, 2, 3]

# Act

num\_cliques, actual\_cliques = method\_subgraph\_handler.find\_cliques(graph, size)

# Assert

self.assertIn(expected\_cliques, actual\_cliques)

def test\_find\_cliques\_\_3\_size\_clique\_and\_5\_size\_graph\_\_num\_cliques\_equal\_1(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2],

4: [5],

5: [4]

}

size = 3

expected\_num\_cliques = 1

# Act

actual\_num\_cliques, actual\_cliques = method\_subgraph\_handler.find\_cliques(graph, size)

# Assert

self.assertEqual(actual\_num\_cliques, expected\_num\_cliques)

def test\_find\_cliques\_\_2\_size\_clique\_and\_3\_size\_graph\_\_num\_cliques\_equal\_3(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

size = 2

expected\_num\_cliques = 3

# Act

actual\_num\_cliques, actual\_cliques = method\_subgraph\_handler.find\_cliques(graph, size)

# Assert

self.assertEqual(actual\_num\_cliques, expected\_num\_cliques)

def test\_find\_cliques\_\_2\_size\_clique\_and\_3\_size\_graph\_\_list\_of\_1\_2\_in\_found\_cliques(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

size = 2

expected\_cliques = [1, 2]

# Act

actual\_num\_cliques, actual\_cliques = method\_subgraph\_handler.find\_cliques(graph, size)

# Assert

self.assertIn(expected\_cliques, actual\_cliques)

def test\_find\_cliques\_\_2\_size\_clique\_and\_3\_size\_graph\_\_list\_of\_1\_3\_in\_found\_cliques(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

size = 2

expected\_cliques = [1, 3]

# Act

actual\_num\_cliques, actual\_cliques = method\_subgraph\_handler.find\_cliques(graph, size)

# Assert

self.assertIn(expected\_cliques, actual\_cliques)

def test\_find\_cliques\_\_2\_size\_clique\_and\_3\_size\_graph\_\_list\_of\_2\_3\_in\_found\_cliques(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

size = 2

expected\_cliques = [2, 3]

# Act

actual\_num\_cliques, actual\_cliques = method\_subgraph\_handler.find\_cliques(graph, size)

# Assert

self.assertIn(expected\_cliques, actual\_cliques)

class TestNegativeFindCliquesMethods(unittest.TestCase):

# Negative

def test\_find\_cliques\_\_2\_size\_clique\_and\_3\_size\_graph\_\_list\_of\_2\_1\_not\_in\_found\_cliques(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

size = 2

expected\_cliques = [2, 1]

# Act

actual\_num\_cliques, actual\_cliques = method\_subgraph\_handler.find\_cliques(graph, size)

# Assert

self.assertNotIn(expected\_cliques, actual\_cliques)

def test\_find\_cliques\_\_2\_size\_clique\_and\_3\_size\_graph\_\_num\_cliques\_not\_equal\_1(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

size = 2

expected\_num\_cliques = 1

# Act

actual\_num\_cliques, actual\_cliques = method\_subgraph\_handler.find\_cliques(graph, size)

# Assert

self.assertNotEqual(actual\_num\_cliques, expected\_num\_cliques)

def test\_find\_cliques\_\_2\_size\_clique\_and\_3\_size\_graph\_\_num\_cliques\_not\_equal\_2(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

size = 2

expected\_num\_cliques = 2

# Act

actual\_num\_cliques, actual\_cliques = method\_subgraph\_handler.find\_cliques(graph, size)

# Assert

self.assertNotEqual(actual\_num\_cliques, expected\_num\_cliques)

class TestPositiveFindSubcliquesMethods(unittest.TestCase):

# Positive

def test\_find\_subcliques\_\_3\_size\_clique\_and\_3\_size\_graph\_and\_prev\_nodes\_list\_of\_1\_2\_and\_nodes\_dict\_of\_3\_\_list\_of\_1\_2\_3\_in\_subcliques(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

prev\_nodes = [1, 2]

nodes = {3}

size = 3

expected\_subclique = [1, 2, 3]

# Act

actual\_subcliques = list(method\_subgraph\_handler.find\_subcliques(graph, prev\_nodes, nodes, size))

# Assert

self.assertIn(expected\_subclique, actual\_subcliques)

def test\_find\_subcliques\_\_3\_size\_clique\_and\_3\_size\_graph\_and\_prev\_nodes\_list\_of\_1\_2\_and\_nodes\_dict\_of\_3\_\_subcliques\_count\_equal\_1(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

prev\_nodes = [1, 2]

nodes = {3}

size = 3

expected\_subcliques\_count = 1

# Act

actual\_subcliques = list(method\_subgraph\_handler.find\_subcliques(graph, prev\_nodes, nodes, size))

# Assert

self.assertEqual(len(actual\_subcliques), expected\_subcliques\_count)

def test\_find\_subcliques\_\_2\_size\_clique\_and\_3\_size\_graph\_and\_prev\_nodes\_list\_of\_1\_and\_nodes\_dict\_of\_2\_3\_\_list\_of\_1\_3\_in\_subcliques(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

prev\_nodes = [1]

nodes = {2, 3}

size = 2

expected\_subcliques = [1, 3]

# Act

actual\_subcliques = list(method\_subgraph\_handler.find\_subcliques(graph, prev\_nodes, nodes, size))

# Assert

self.assertIn(expected\_subcliques, actual\_subcliques)

def test\_find\_subcliques\_\_2\_size\_clique\_and\_3\_size\_graph\_and\_prev\_nodes\_list\_of\_1\_and\_nodes\_dict\_of\_2\_3\_\_list\_of\_1\_2\_in\_subcliques(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

prev\_nodes = [1]

nodes = {2, 3}

size = 2

expected\_subcliques = [1, 2]

# Act

actual\_subcliques = list(method\_subgraph\_handler.find\_subcliques(graph, prev\_nodes, nodes, size))

# Assert

self.assertIn(expected\_subcliques, actual\_subcliques)

def test\_find\_subcliques\_\_3\_size\_clique\_and\_2\_size\_graph\_and\_prev\_nodes\_list\_of\_1\_and\_nodes\_dict\_of\_2\_3\_\_subcliques\_count\_equal\_2(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

prev\_nodes = [1]

nodes = {2, 3}

size = 2

expected\_subcliques\_count = 2

# Act

actual\_subcliques = list(method\_subgraph\_handler.find\_subcliques(graph, prev\_nodes, nodes, size))

# Assert

self.assertEqual(len(actual\_subcliques), expected\_subcliques\_count)

class TestNegativeFindSubcliquesMethods(unittest.TestCase):

# Negative

def test\_find\_subcliques\_\_2\_size\_clique\_and\_3\_size\_graph\_and\_prev\_nodes\_list\_of\_1\_and\_nodes\_dict\_of\_2\_3\_\_list\_of\_2\_1\_not\_in\_subcliques(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

prev\_nodes = [1]

nodes = {2, 3}

size = 2

expected\_subcliques = [2, 1]

# Act

actual\_subcliques = list(method\_subgraph\_handler.find\_subcliques(graph, prev\_nodes, nodes, size))

# Assert

self.assertNotIn(expected\_subcliques, actual\_subcliques)

def test\_find\_subcliques\_\_3\_size\_clique\_and\_2\_size\_graph\_and\_prev\_nodes\_list\_of\_1\_and\_nodes\_dict\_of\_2\_3\_\_subcliques\_count\_not\_equal\_1(self):

# Arrange

graph = {

1: [2, 3],

2: [1, 3],

3: [1, 2]

}

prev\_nodes = [1]

nodes = {2, 3}

size = 2

expected\_subcliques\_count = 1

# Act

actual\_subcliques = list(method\_subgraph\_handler.find\_subcliques(graph, prev\_nodes, nodes, size))

# Assert

self.assertNotEqual(len(actual\_subcliques), expected\_subcliques\_count)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()