Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления
Кафедра Интеллектуальных информационных технологий

К защите допустить:
Заведующий кафедрой
_____ Д. В. Шункевич

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к расчетной работе
по дисциплине «Проектирование программ в интеллектуальных системах»:

Большое модульное произведение графов

БГУИР РР 1–40 03 01 49

Студент: Д. Н. Михалькевич
Группа: 221701

Руководитель: С. А. Никифоров

Минск 2023

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

БЗ — база знаний;
SC — Semantic Code;
SCg —Semantic Code Graphical;
SCn — Semantic Code Natural;
it3 — Итератор для 2 узлов и связи между ними;
it5 — Итератор для 2 узлов и 1 узла отношения и связей между ними.

ВВЕДЕНИЕ

Цель: Создать агента для веб платформы OSTIS, который производит операцию большого модульного произведения двух неориентированных графов.

Задача: Найти большое модульное произведение двух неориентированных графов.

1 ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

– **Граф** — совокупность двух множеств: множества самих объектов, называемого множеством вершин (V), и множества их парных связей, называемого множеством рёбер (E);

– **Неориентированный граф** – граф, в котором рёбра не имеют направления.

– **Большое модульное произведение**[1] $G_1 \Diamond G_2$ д**двух неориентированных графов** содержит рёбра $((x^1, x^2), (y^1, y^2)) \in E(G_1 \Diamond G_2)$ тогда и только тогда, когда $x^1 \neq y^1, x^2 \neq y^2$ и либо $(x^1, y^1) \in E_1$ и $(x^2, y^2) \in E_2$, либо $(x^1, y^1) \notin E_1$, где x^1 и $y^1 \in V_1$, x^2 и $y^2 \in V_2$, E_1 – множество ребер первого графа, E_2 – множество ребер второго графа, V_1 – множество вершин первого графа, V_2 – множество вершин первого графа.

2 АЛГОРИТМ БОЛЬШОГО МОДУЛЬНОГО ПРОИЗВЕДЕНИЯ

2.1 Описание алгоритма

Алгоритм построения графа большого модульного произведения двух графов следующий:

1) Создаём пустой граф, который будет являться результатом большого модульного произведения.

2) Создаём пустое множество вершин графа большого модульного произведения.

3) Производим операцию декартова произведения множеств вершин первого и второго графа. Результатом этой операции и будет множество вершин большого модульного произведения.

4) Создаём множество рёбер графа большого модульного произведения.

5) Заполняем его рёбрами по условию:

ребро $((x^1, x^2), (y^1, y^2)) \in E(G_1 \Diamond G_2)$ тогда и только тогда, когда $x^1 \neq y^1, x^2 \neq y^2$ и либо $(x^1, y^1) \in E_1$ и $(x^2, y^2) \in E_2$, либо $(x^1, y^1) \notin E_1$,

где x^1 и $y^1 \in V_1$, x^2 и $y^2 \in V_2$, E_1 – множество ребер первого графа, E_2 – множество ребер второго графа, V_1 – множество вершин первого графа, V_2 – множество вершин первого графа.

6) Из множества вершин и множества рёбер строим граф-результат.

7) Граф результат — большое модульное произведение.

2.2 Реализация алгоритма для агента

В качестве входных данных агенту приходит 2 графа.

Затем агент начинает работу согласно алгоритму, а именно создаёт структуру ответа и последовательно заносит исходные графы в неё.

В начале агент принимает входные графы через $OtherAddr$ в $ActionNode$ и создаёт вектор адресов $Input$, в который при помощи $it3$ заносит все элементы входных графов. После создаёт ещё один вектор адресов $newEdges$ уже для хранения результата работы агента.

Чтобы декомпозировать код агента созданы функции:

– $createResponseConstruction$ (функция составления структуры ответа)

– $processing$ (функция обработки входных графов в вектор структур, содержащий множество вершин графа большого модульного произведения)

– $detectEdges$ (функция определения рёбер графа большого модульного произведения)

– *addResultEdges* (функция создания ребра), вызываемая из функции *detectEdges*.

Хранение вершин графа большого модульного произведения реализовано в структуре $FinalNode$ с полями:

– $firstcolor$ — адрес вершины первого графа;

– *secondcolor* — адрес вершины второго графа;

– *name* — строка, имя вершины графа большого модульного произведения.

Так как во входных графах есть вершины без связей (рёбер), эти "пустые"ребра заменены ребрами типа $EdgeUCommonVar$ для удобства и простоты работы агента.

2.2.1 *createResponseConstruction* служит для составления структуры ответа. Её аргументами являются: *newEdges* (вектор адресов ответа), $graph1$ (первый исходный граф), $graph2$ (второй исходный граф). В начале создаётся вектор структур *newNodes* (служит для хранения множества вершин графа большого модульного произведения). И вызываются функции *processing* и *detectEdges*.

2.2.2 *processing* с помощью $it3$ проходит по вершинам первого графа, для каждой его вершины, вызываем ещё один $ti3$, который проходит по вершинам второго графа, создавая множество вершин и соответственно заполняя вектор структур вершин графа большого модульного произведения. Данные вершин содержатся в *newNodes*.

2.2.3 *detectEdges* — это функция определения рёбер графа большого модульного произведения.

Функция проходит по всем вершинам графа большого модульного произведения (*newNodes*) с помощью вложенного в цикл for по i цикл for по j, заполненного в прошлой функции. Во внутреннем цикле for проверяется не существование циклов для $graph1$ и $graph2$, то есть, что в выбранных двух вершинах части идентификаторов для соответствующих двух вершин входных графов не одинаковы.

Затем с помощью $it5$ мы проверяем условие существования ребра между выбранными вершинами:

Определяем, наличие ребра для соответствующих вершин в первом входном графе ($graph1$).

Для определения того, есть ли неориентированная дуга (типа $EdgeUCommonConst$) или ее нет используется переменная $counter = 0$, если такая дуга обнаружена при проверке первого входного графа, описанной выше, то *counter* станет равным 1.

Если ребро в первом графе есть ($counter = 1$), то проверяется с помощью $it5$, если ребро для соответствующих вершин во втором входном графе ($graph2$) . Если ребро есть во втором графе, то вызывается функция $addResultEdges$ для добавления ребра для этих вершин (вершины i и j вектора $newNodes$).

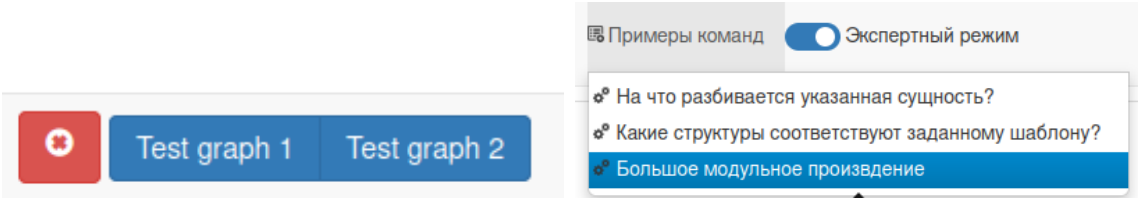
Если же ребра для соответствующих вершин в первом графе нет ($counter$ остался равным нулю), то проверяется есть ли "ребро-пустышка"($EdgeUCommonVar$) между соответствующими вершинами в первом графе с помощью $it5$, ели оно есть, то вызывается функция $addResultEdges$ для добавления ребра для этих вершин (вершины i и j вектора $newNodes$).

В функции дважды проверяется есть ли ребро (между вершинами i, j и j, i) потому, что мы имеем дело с неориентированным графом, поэтому из-за особенностей построения графа в kbe необходимо проверить путь ребра из узла 1 в узел 2 и наоборот, так как мы не знаем, от какого узла к какому строились рёбра в исходных графах.

2.2.4 $addResultEdges$ — функция добавления рёбер. Функция создаёт 2 вершины и методом $HelperResolveSystemIdtf$ присваивает им идентификаторы: создаёт или берёт идентификатор из узлов ответа (если такие узлы уже были созданы) и создает между ними ребро методом $CreateEdge$. Затем добавляет это ребро в структуру ответа $newEdges$.

3 ПРИМЕРЫ И РЕЗУЛЬТАТЫ РАБОТЫ АГЕНТА

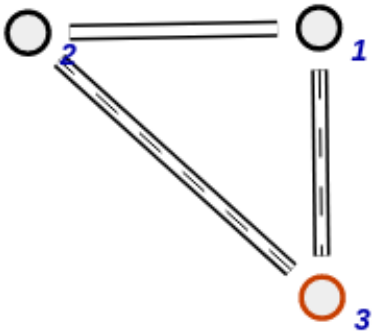
Чтобы вызвать агента нужно закрепить два исходных графа, из которых мы хотим посчитать граф большого модульного произведения, используя инструмент “булавка“. После чего вызвать агента через меню "Примеры команд"и нажать на "Большое модульное произведение".



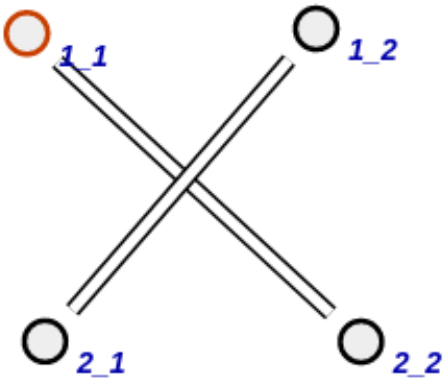
(а) Закреплённые исходные графы (б) Агент в выборе команд

Рисунок *3.1*

3.1 Пример 1

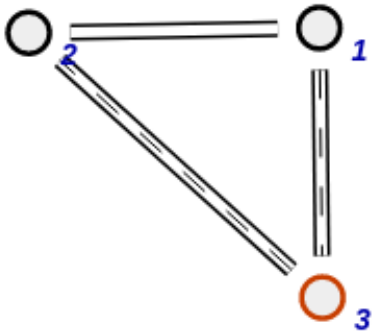


(а) Входной граф 1 (б) Входной граф 2

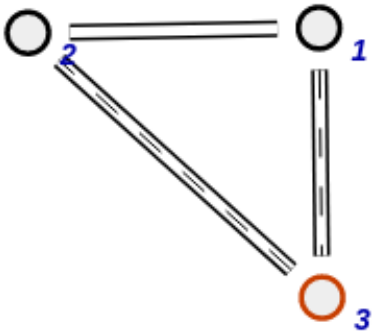


(в) Результат работы агента

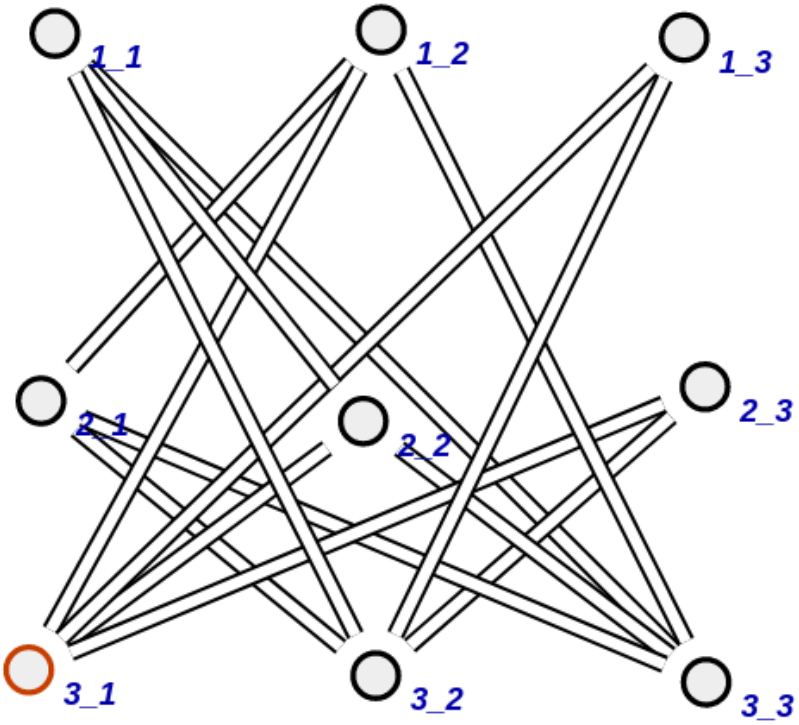
3.2 Пример 2.1



(а) Входной граф 1

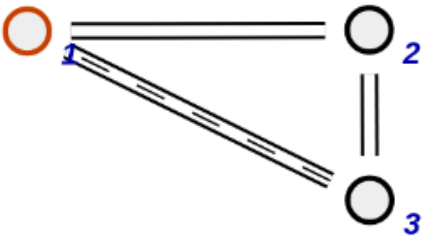


(б) Входной граф 2

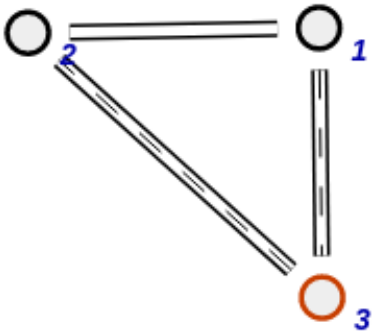


(в) Результат работы агента

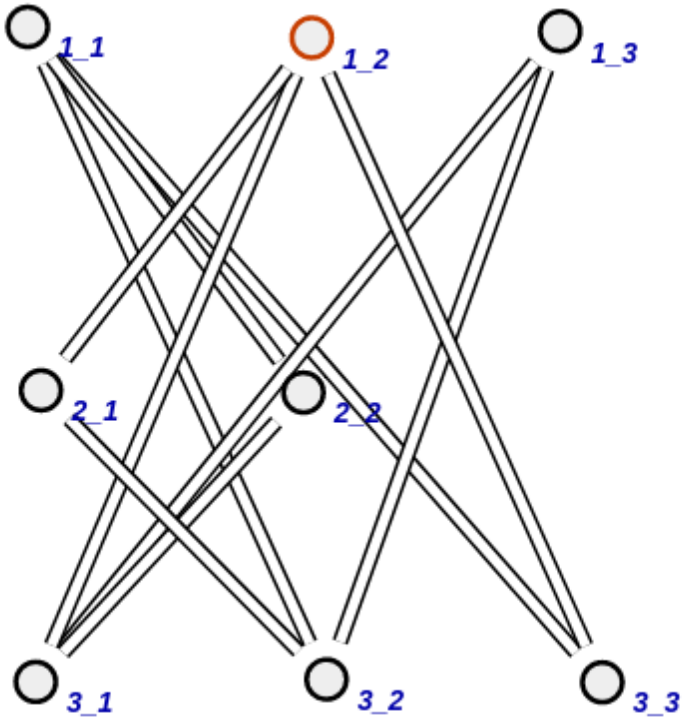
3.3 Пример 3



(а) Входной граф 1

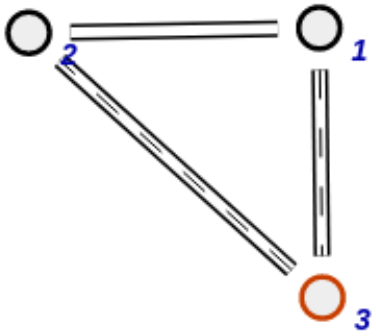


(б) Входной граф 2

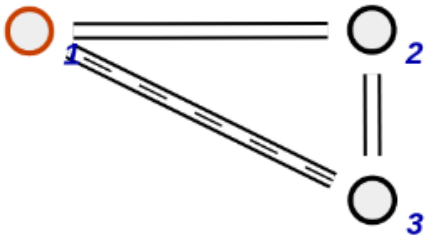


(в) Результат работы агента

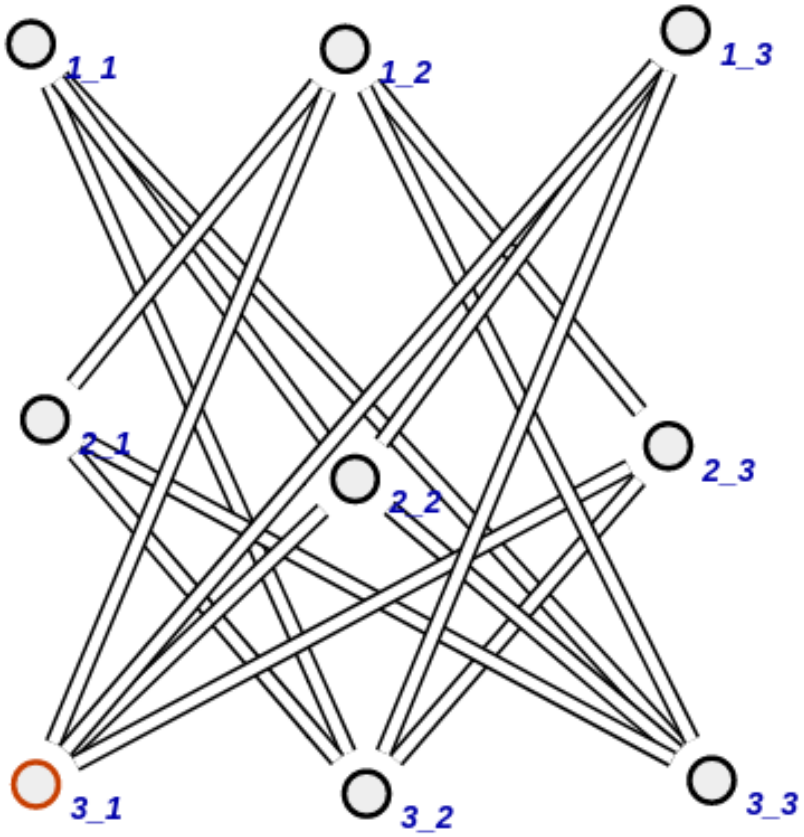
3.4 Пример 4



(а) Входной граф 1

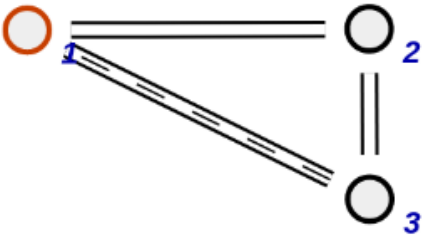


(б) Входной граф 2

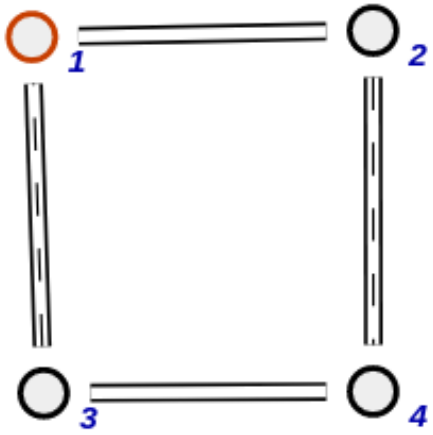


(в) Результат работы агента

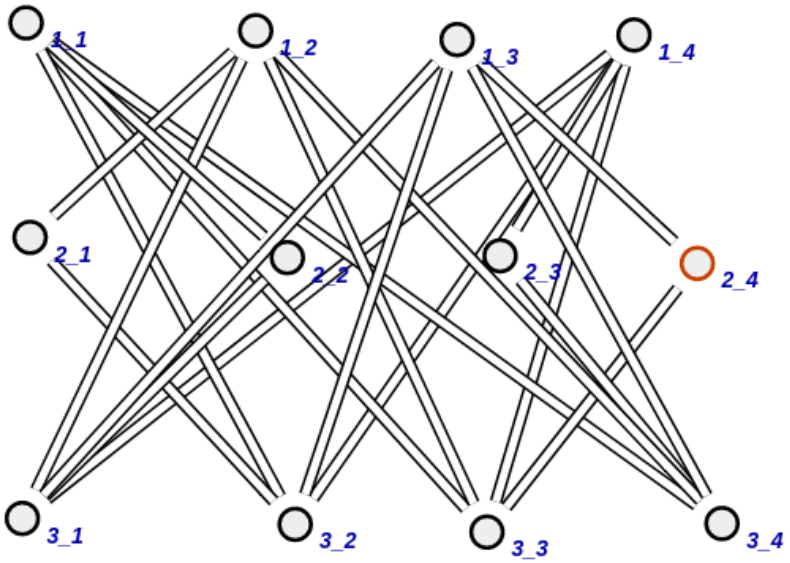
3.5 Пример 5



(а) Входной граф 1

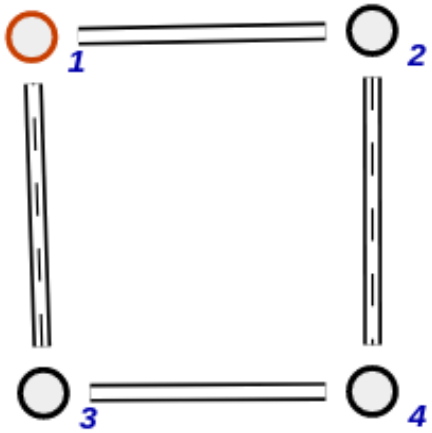


(б) Входной граф 2

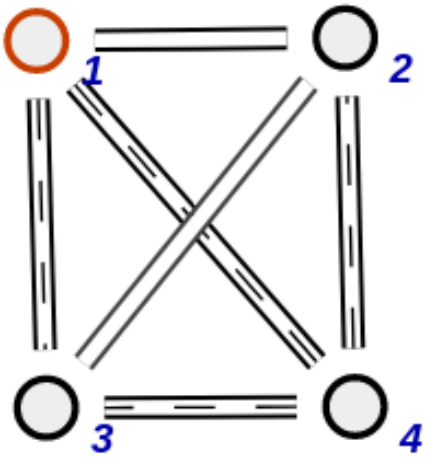


(в) Результат работы агента

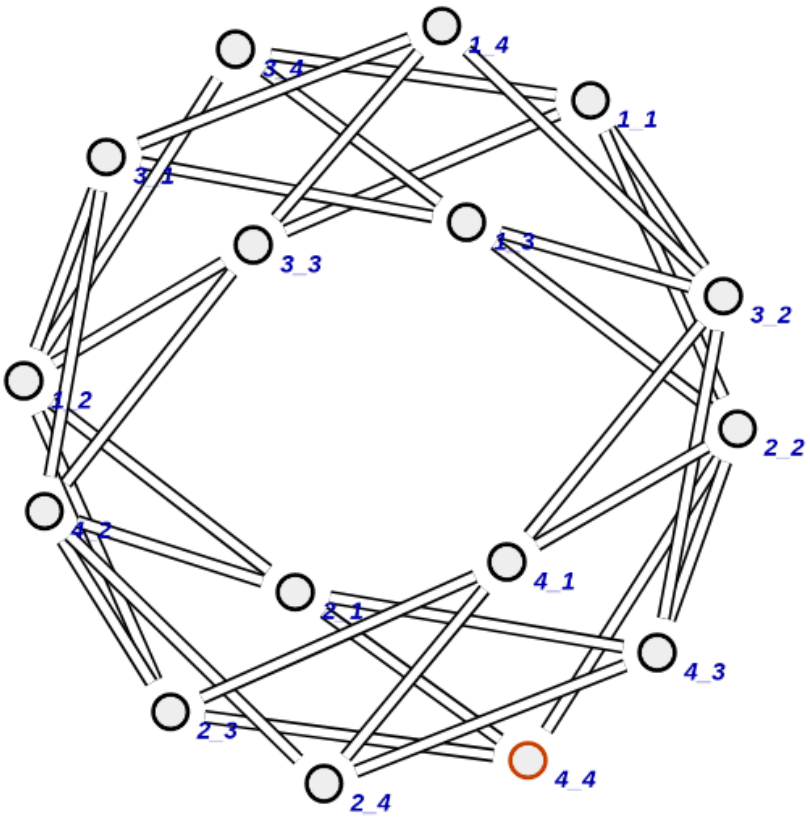
3.6 Пример 6



(а) Входной граф 1



(б) Входной граф 2



(в) Результат работы агента

ЗАКЛЮЧЕНИЕ

Выполняя данную расчётную работу я изучил то, как работают агенты в экосистеме, и также написал своего агента, выполняющего операцию большого модульного произведения графов. Повторил и улучшил свои знания теории графов и реализации алгоритмов на графах разной сложности. Изучил и применил различные методы взаимодействия с системой OSTIS. Усовершенствовал свои навыки проектирования программных средств на языке программирования C++.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] WikiGRAPP. https://shorturl.at/jHXY2.

[2] Документация программного варианта реализации платформы интерпретации компьютерных систем. https://github.com/ostis-ai/ostis-web-platform/blob/develop/docs/.

[3] Документация программного варианта реализации платформы интерпретации компьютерных систем. https://github.com/Mediano4e/bsuir_guides/tree/main/example_app_agent.