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 программного обеспечения интеллектуальных систем»:

Регулярный, реберно-регулярный неориентированный граф

Студент: В. Д. Семеняко
Группа: 321701

Руководитель: М. Е. Садовский

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Графовые структуры играют важную роль в моделировании различных систем и процессов, от социальных сетей и интернет-маршрутизации до биологических и транспортных сетей. Одной из ключевых задач, связанных с графами, является проверка их регулярности, то есть определение, имеют ли все вершины одинаковую степень. Анализ регулярности графа помогает лучше понимать его структуру, выявлять закономерности и использовать эти знания для оптимизации сетей, моделирования симметричных систем и оценки их надежности и устойчивости.

Цель работы — разработка агента для определения, является ли заданный граф регулярным. Агент должен автоматически анализировать степень каждой вершины графа и проверять, одинаковы ли они для всех вершин. На основе этого анализа агент определяет, является ли граф регулярным, и возвращает соответствующий результат.

Для достижения поставленной цели необходимо решить следующие задачи:

– Разработать алгоритм, который будет эффективно извлекать все вершины и их степени из исходного графа.

– Реализовать механизм для проверки равенства степеней всех вершин графа, чтобы агент мог определить его регулярность.

– Обеспечить вывод результата проверки регулярности в удобном формате для дальнейшего анализа.

– Организовать формирование структурированного отчета, пригодного для использования в других задачах и приложениях.

Актуальность работы обусловлена широким применением графов и их свойств в таких сферах, как физика, химия, экономика и инженерия. Способность автоматически определять, является ли граф регулярным, открывает возможности для анализа симметричных структур, моделирования равномерных распределений в сетях и оценки стабильности систем. Разработка агента для решения этой задачи поможет ускорить исследование графов, минимизировать вероятность ошибок при ручной проверке и упростить работу с большими массивами данных, что особенно важно в задачах промышленного и научного моделирования.

1 СПИСОК ИСПОЛЬЗОВАННЫХ ПОНЯТИЙ

а) ***Неориентированный граф*** (абсолютное понятие) - граф, в котором все ребра являются звеньями, то есть порядок двух концов ребра графа не существенен

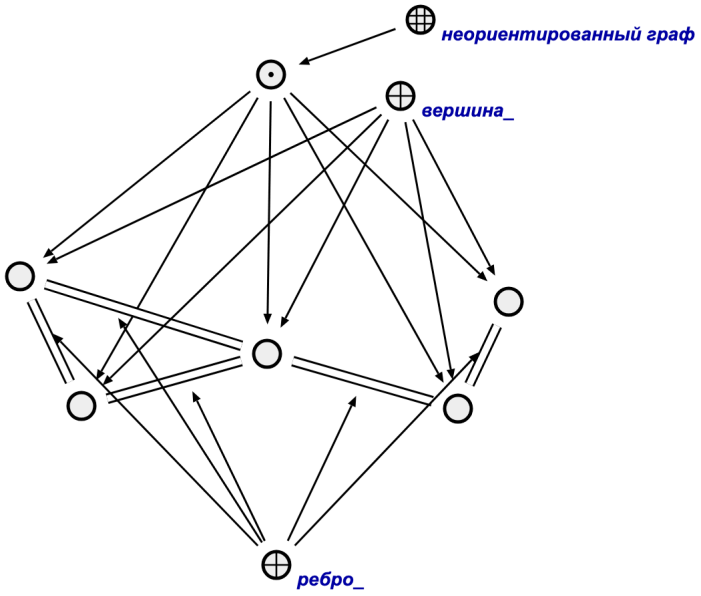

Рисунок *1.1* – Абсолютное понятие неориентированного графа

б) ***Регулярный граф*** (абсолютное понятие) - граф, в котором каждая вершина имеет одинаковое количество соседей, т.е. каждая вершина имеет одинаковую степень или валентность.

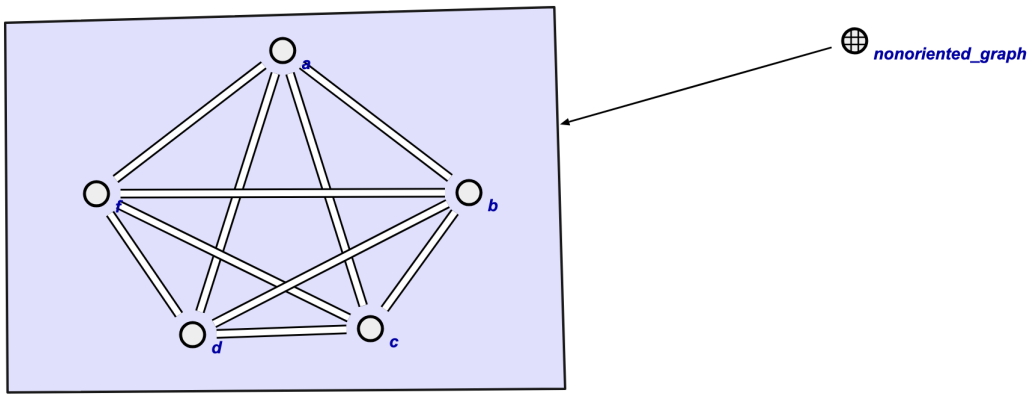

Рисунок *1.2* – Пример регулярного графа

2 АНАЛИТИКА

Для решения задачи определения регулярности графа можно применять различные подходы. В данной работе был разработан агент, реализующий алгоритм проверки степени вершин графа. Алгоритм основывается на последовательном обходе всех вершин и сравнении их степеней для выявления несоответствий. Для реализации использован подход с использованием структур данных для хранения вершин. Такой метод обеспечивает высокую производительность, простоту реализации и минимальные требования к вычислительным ресурсам, что делает его эффективным при работе с графами любых размеров.

Идея алгоритма заключается в следующем: агент собирает все вершины исходного графа в массив и определяет степень каждой вершины. Затем выполняется последовательная проверка на равенство степеней всех вершин графа. Для подсчета степени используется вспомогательная функция, которая эффективно определяет количество соединений каждой вершины. Этот подход позволяет быстро выявить любые несоответствия, минимизируя избыточные операции и обеспечивая точное определение регулярности графа.

Недостатком метода является увеличение времени обработки для графов с большим числом вершин, так как алгоритм проверяет степени каждой вершины, что может быть ресурсоёмким при работе с крупными сетями. Однако данный подход остаётся достаточно эффективным для графов средней сложности и гарантирует точный результат, позволяя надёжно определить, является ли граф регулярным.

3 РАЗРАБОТКА

3.1 Переменные

Основные элементы, использующиеся в алгоритме, это:

– **mcontext** — объект, представляющий контекст, с помощью которого агент взаимодействует с памятью и выполняет операции с элементами графа.

– **degree** — это переменная, которая используется для хранения степени первой вершины, встреченной в процессе обхода графа. Изначально она инициализируется значением -1, чтобы сигнализировать о том, что степень еще не была определена. После нахождения первой вершины, ее степень сохраняется в **degree**. В дальнейшем, для каждой последующей вершины, степень сравнивается с ранее сохраненной. Это значение служит для проверки, все ли вершины графа имеют одинаковую степень.

– **isregular** — булевая переменная, которая отвечает за хранение результата проверки регулярности графа. Изначально она установлена в true, предполагая, что граф регулярный.

3.2 Алгоритм

Функция DoProgram - является основной функцией агента, которая реализует алгоритм идентификации регулярного графа. Она принимает в качестве аргумента *ScAction & action*, в котором содержится информация о графе, который будет исследоваться на регулярность.

Шаги функции DoProgram:

а) **Получение аргумента *graph***. Агент получает аргумент *graph* из действия *action*, который представляет собой адрес графа, для которого нужно провести проверку на регулярность.

б) **Проверка существования графа**. Агент проверяет, существует ли указанный граф в памяти. Для этого используется метод `graph.IsValid()`. Если граф не существует, агент завершает выполнение с ошибкой, выводя сообщение об ошибке через `SC_AGENT_LOG_ERROR` и завершает работу с помощью `action.FinishWithError()`.

в) **Создание итератора для обхода графа**. Агент создаёт итератор `ScIterator3`, который используется для поиска всех элементов графа типа `ScType::EdgeAccessConstPosPerm` и `ScType::NodeConst`.

г) **Инициализация переменных для проверки регулярности графа**. Агент инициализирует переменные `degree` со значением `-1` и `is_regular` со значением `true`. Эти переменные будут использоваться для проверки, является ли граф регулярным.

д) **Цикл по всем вершинам графа**. Агент запускает цикл, в кото-

ром для каждой вершины графа:

1) Извлекает текущую вершину `current_node` с помощью метода `i3->Get(2)`.

2) Пропускает вершину с системным идентификатором `sc_node`, если она найдена.

3) Вычисляет степень текущей вершины с помощью функции `calculateDegree(current_node)`.

4) Если `degree` ещё не инициализировано, присваивает его значение текущей степени. Если степень текущей вершины не совпадает с предыдущей, устанавливает `is_regular` в `false` и завершает цикл.

е) **Построение структуры**. Создаётся структура `ScStructure`, которая будет использоваться для формирования результата.

ж) **Установка результата проверки регулярности**. Если граф является регулярным (`is_regular` равно `true`), то агент ищет элемент с системным идентификатором `concept_regular_graph` и присваивает переменной (`result_node` соответствующее значение. В противном случае агент ищет элемент `concept_irregular_graph` и, аналогично, присваивает переменной (`result_node` соответствующее значение.

з) **Создание дуги для результата**. Агент генерирует дугу `arcAddr` типа `EdgeAccessConstPosPerm` между элементами графа и результатом, добавляет все элементы в структуру `structure` и устанавливает результат действия с помощью `action.SetResult(structure)`.

и) **Завершение выполнения**. Агент завершает выполнение действия с помощью `action.FinishSuccessfully()`.

Функция calculateDegree - Вспомогательная функция, которая вычисляет степень заданной вершины заданного графа. Она принимает на вход вершину ScAddr node, степень которой будет вычисляться.

Шаги функции calculateDegree:

а) **Инициализация переменной** `degree`. Функция начинает с присваивания переменной `degree`, значение `0`.

б) **Создание итератора для обхода рёбер вершины**. С помощью метода `m_context.CreateIterator3` создаётся итератор `edgeIterator`, который используется для обхода рёбер, исходящих из вершины `node`. Итератор ищет рёбра типа `ScType::EdgeUCommonConst`, которые соединяют вершину `node` с другими вершинами, и находит связанные с ними элементы типа `ScType::NodeConst`.

в) **Цикл по рёбрам вершины**. В цикле `while (edgeIterator->Next())`, агент перебирает все рёбра, исходящие из вершины `node`. Каждый раз, когда итератор находит ребро, переменная `degree` увеличивается на единицу.

г) **Возвращение результата**. После завершения цикла, функция возвращает значение переменной `degree`, которое представляет собой сте-

пень вершины, то есть количество рёбер, исходящих из этой вершины.

4 ПОКАЗ ТЕСТОВЫХ СЦЕНАРИЕВ

а) **Тест 1:**

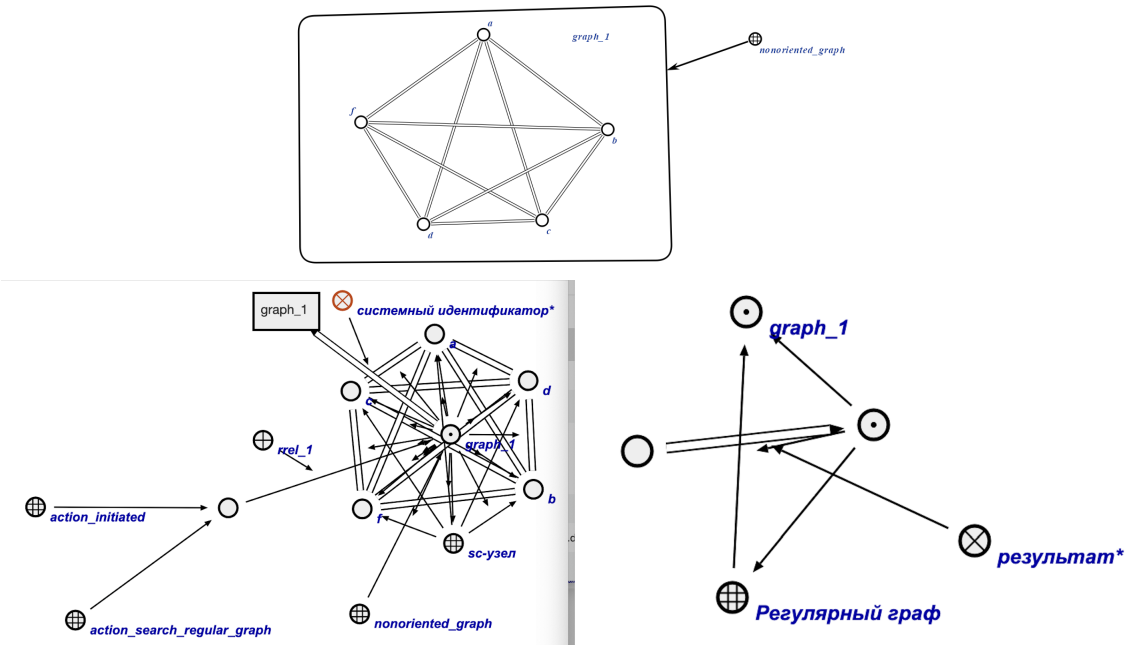

Рисунок *4.1* – Идентификация графа на регулярность

б) **Тест 2:**

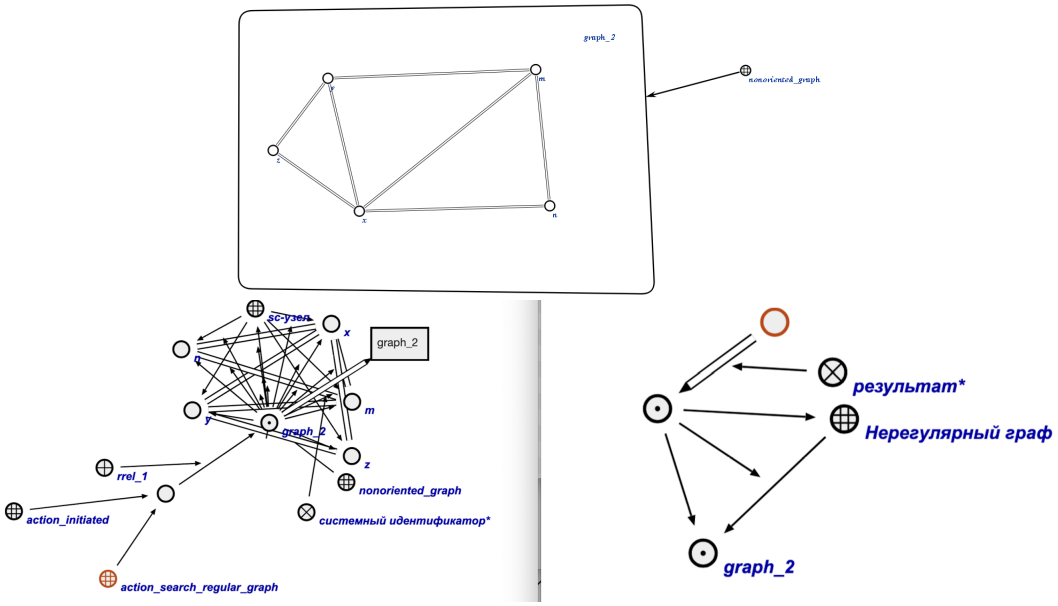

Рисунок *4.2* – Идентификация графа на регулярность

в) **Тест 3:**

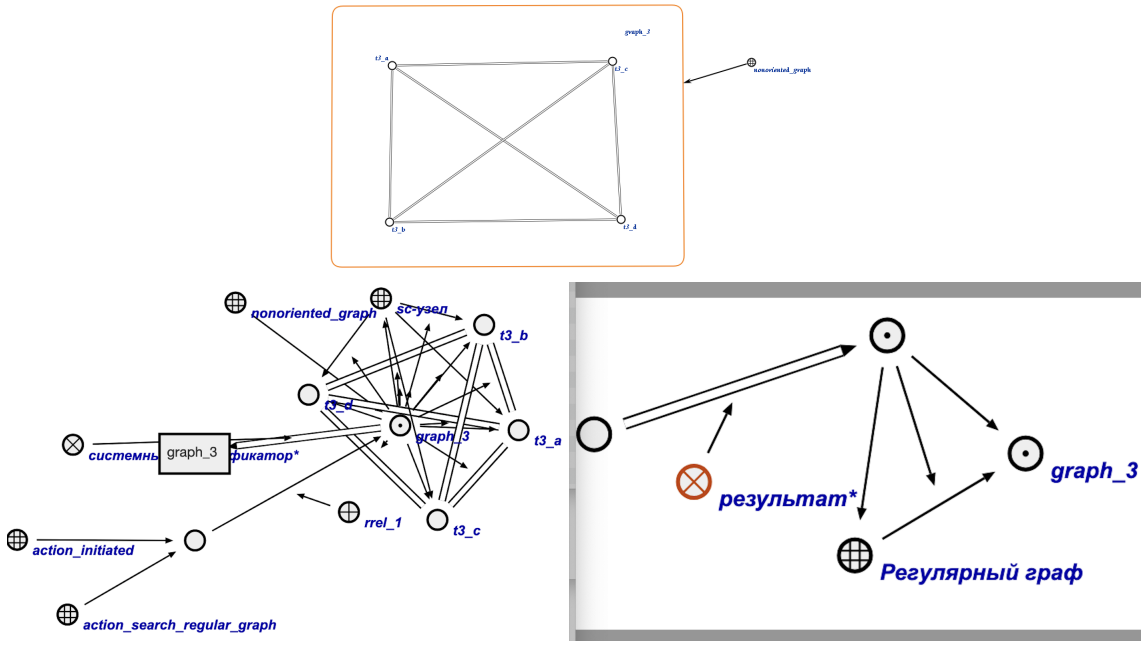

Рисунок *4.3* – Идентификация графа на регулярность

г) **Тест 4:**

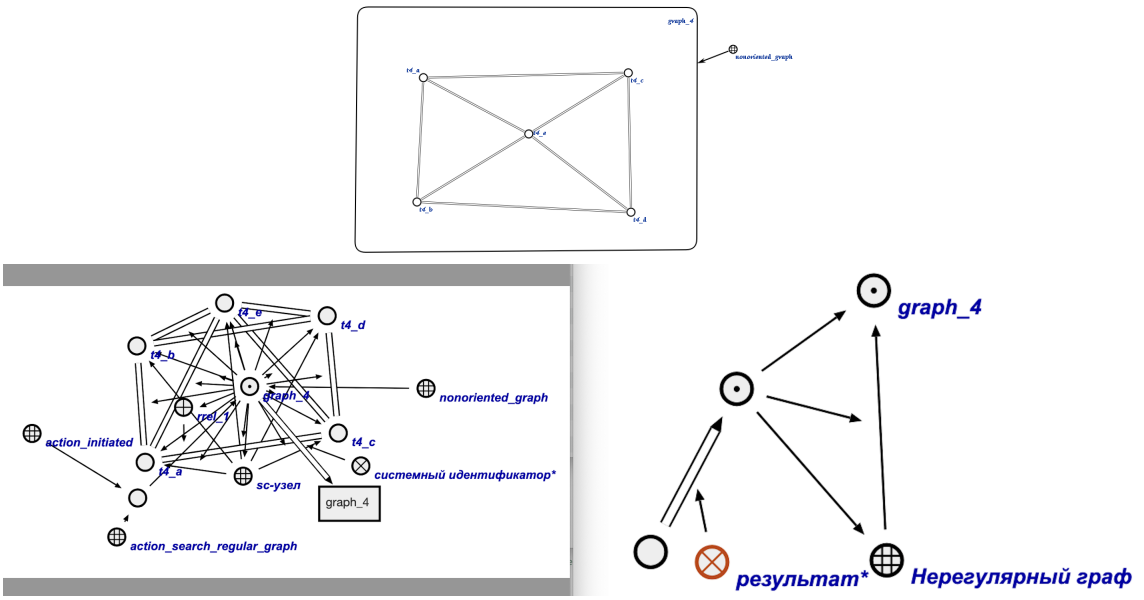

Рисунок *4.4* – Идентификация графа на регулярность

д) **Тест 5:**

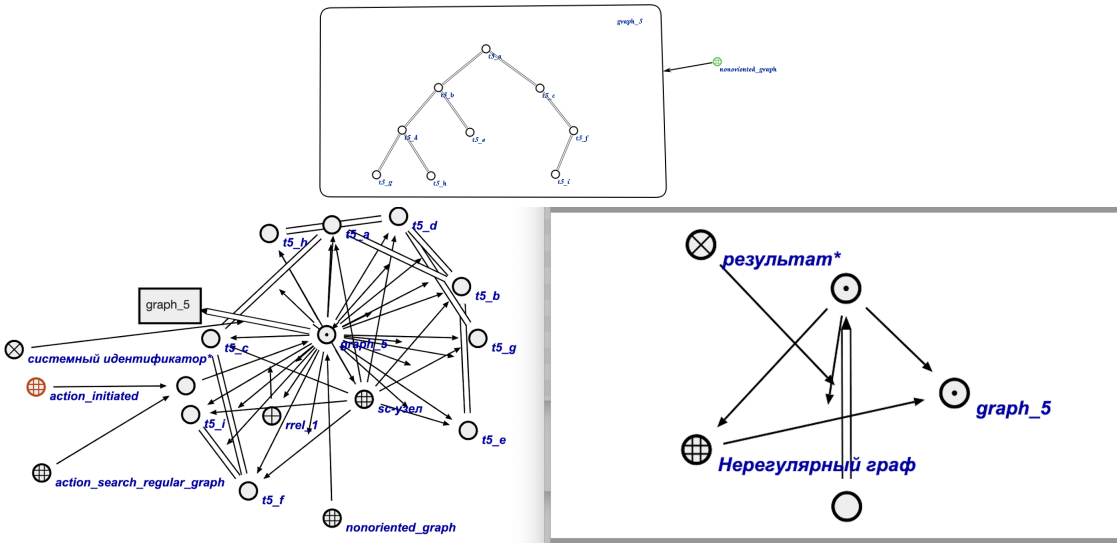

Рисунок *4.5* – Идентификация графа на регулярность

ЗАКЛЮЧЕНИЕ

В рамках работы был разработан агент, предназначенный для автоматической проверки, является ли граф регулярным. Этот инструмент предоставляет возможность анализировать структуру графа и выявлять, есть ли в нем вершины с одинаковым числом связей. Алгоритм, на основе которого работает агент, позволяет эффективно обходить все вершины графа, сравнивать их степени и устанавливать, удовлетворяет ли граф свойству регулярности.

Работа агента опирается на тщательно продуманную структуру и использование эффективных методов обхода графа, таких как итераторы для анализа вершин и определения их степени. Этот процесс приводит к тому, что агент может точно и быстро определить, является ли граф регулярным, в случае чего выдается соответствующий результат.

Разработка и внедрение этого агента имеет практическую значимость для ряда задач, связанных с теорией графов. Определение регулярности графа находит применение в различных областях, включая анализ социальных сетей, где важно учитывать степень взаимосвязанности узлов, а также в задаче оптимизации сетевых структур, где знание регулярности может повлиять на эффективность работы сети.

Таким образом, агент для проверки регулярности графа представляет собой не только решение конкретной задачи, но и важный шаг в направлении создания более универсальных инструментов для анализа графовых структур. Разработка этого инструмента открывает новые перспективы для применения в теоретических и прикладных задачах, где знание о регулярности графа может помочь глубже понять его свойства и поведение.

Подводя итог, можно заключить, что поставленные цели были успешно достигнуты. Агент был реализован и протестирован, что подтверждает его работоспособность и эффективность. Результаты работы открывают возможности для дальнейшего использования этого инструмента в научных и практических исследованиях, связанных с анализом графов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Харари, Фрэнк. Теория графов / Фрэнк Харари. — 1973.

[2] Иванова, ГС. Полная характеристика структуры неориентированного графа / ГС Иванова, ВА Овчинников. No. 4. — Некоммерческое партнерство «Национальный Электронно-Информационный Консорциум», 2016. — Pp. 106–123.

[3] Фон-Дер-Флаас, Дмитрий Германович. Локальные дополнения простых и ориентированных графов / Дмитрий Германович Фон-Дер-Флаас // Дискретный анализ и исследование операций. — 1994. — Vol. 1, no. 1. — Pp. 43–62.

[4] sc-machine. `https://ostis-ai.github.io/sc-machine/`.

А ЛИСТИНГ КОДА

```
#include "search_regular_graph.hpp"
#include "keynodes/search_keynodes.hpp"

ScAddr SearchRegularGraphAgent::GetActionClass() const {
    return SearchKeynodes::action_search_regular_graph;
}

ScResult SearchRegularGraphAgent::
    DoProgram(ScAction & action)
    {
    auto const & [graph] = action.GetArguments<1>();
    if (!graph.IsValid()) {
        SC_AGENT_LOG_ERROR
        ("Graph node argument is not found");
        return action.FinishWithError();
    }
    ScIterator3Ptr i3 = m_context.CreateIterator3(graph,
    ScType::EdgeAccessConstPosPerm,
    ScType::NodeConst);
    int degree = -1;
    bool is_regular = true;

    while (i3->Next()) {
        ScAddr current_node = i3->Get(2);
        if (current_node == m_context.
        SearchElementBySystemIdentifier(
        "sc_node"
        )) {
            continue;
        }
        int current_degree = calculateDegree
        (current_node);
        if (degree == -1){
            degree = current_degree;
        } else if (degree != current_degree) {
            is_regular = false;
            break;
        }
    }
```

```
    ScStructure structure = m_context.GenerateStructure();
    ScAddr result_node;

    if (is_regular) {
        result_node = m_context.
        SearchElementBySystemIdentifier
        ("concept_regular_graph");
        SC_LOG_DEBUG("Graph is regular.");
    } else {
        result_node = m_context.
        SearchElementBySystemIdentifier
        ("concept_irregular_graph");
        SC_LOG_DEBUG("Graph is not regular.");
    }

    ScAddr const& arcAddr = m_context.GenerateConnector(
    ScType::EdgeAccessConstPosPerm,
    result_node, graph);
    structure << graph << result_node << arcAddr;
    action.SetResult(structure);
    return action.FinishSuccessfully();
}

int SearchRegularGraphAgent::calculateDegree(ScAddr node) {
    int degree = 0;
    ScIterator3Ptr edgeIterator = m_context.
    CreateIterator3(node,
    ScType::EdgeUCommonConst, ScType::NodeConst);
    while (edgeIterator->Next()) {
        degree++;
    }

    return degree;
}
```