Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехники и комплексной автоматизации»

КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Методы оптимизации»

на тему

«Решение задачи поиска циклов в ориетированном графе»

Студент РК6-72Б _____ Ершов В.А.
группа подпись, дата ФИО

Руководитель КП _____ Соколов А.П.
подпись, дата ФИО

Консультант _____ Першин А.Ю.
подпись, дата ФИО

Москва, 2021

Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой *РК-6*
индекс

_____ *А.П. Карпенко*

«____» _____ 2021 г.

ЗАДАНИЕ

на выполнение курсового проекта

Студент группы: РК6-72Б

Ершов Виталий Алексеевич

(фамилия, имя, отчество)

Тема курсового проекта : Решение задачи поиска циклов в ориетированном графе

Источник тематики (кафедра, предприятие, НИР): кафедра

Тема курсового проекта утверждена на заседании кафедры «Системы автоматизированного проектирования (РК-6)», Протокол № _____ от «____» _____ 2021 г.

Техническое задание

Часть 1. *Аналитический обзор литературы.*

В рамках аналитического обзора литературы необходимо проанализировать актуальность исследований в области оптимизации процесса реализации вычислительных методов, найти существующие программные инструменты, позволяющие оптимизировать процесс разработки. Должны быть определены перспективы использования графоориентированного подхода для реализации вычислительных методов.

Часть 2. *Математическая постановка задачи, разработка архитектуры программной реализации, программная реализация.*

Необходимо описать архитектуру разрабатываемого редактора графов, разработать web-ориетированного приложение, позволяющее создавать графовые модели вычислительных методов.

Часть 3. *Проведение вычислительных экспериментов, отладка и тестирование.*

В рамках тестирования необходимо представить ряд примеров, показывающих реализованные в редакторе функциональные возможности.

Оформление курсового проекта :

Расчетно-пояснительная записка на 19 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):

количество: 6 рис., 0 табл., 9 источн.
[здесь следует ввести количество чертежей, плакатов]

Дата выдачи задания «*26*» сентября 2021 г.

Студент _____ Ершов В.А.
подпись, дата ФИО

Руководитель курсового проекта _____ Соколов А.П.
подпись, дата ФИО

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

РЕФЕРАТ

Данная работа посвящена разработке web-ориентированного редактора графов, который является очень полезным инструментом при использовании графоориентированного программного карскаса для реализации сложных вычислительных методов, представленного в работе [1]. Разрабатываемый редактор позволяет создавать графовые модели в соответствии с рассматриваемым графоориентированным подходом, экспортировать созданные графы в формате aDOT, а также загружать заранее подготовленные графовые модели из этого формата. В работе описана важность данной разработки, представлена архитектура разрабатываемого редактора и рассмотрены реализованные в редакторе функциональные возможности, а также приведен ряд примеров, демонстрирующих его работу.

Тип работы: курсовой проект .

Тема работы: *«Решение задачи поиска циклов в ориетированном графе»*.

Объект исследования: Применение графориентированного подхода для реализации сложных вычислительных методов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

При разработке любого программного обеспечения разработчики ставят перед собой несколько очень важных задач: эффективность и гибкость дальнейшей поддержки программного обеспечения. В современной разработке существует множество паттернов проектирования, архитектур и вспомогательных систем, которые позволяют быстрее создавать программное обеспечения и в дальнейшем упрощают его поддержку для новых разработчиков. Таким образом, даже в небольших проектах, над которыми работает небольшая команда разработчиков стараются использовать системы контроля версий, юнит-тестирование. Если приложение представляет из себя API при его разработке используют специальные системы для быстрой документации API. Такие подходы считаются стандартом при разработке программного обеспечения рассчитанное на удовлетворение потребностей конечных пользователей - разработка мобильных приложений, проектирование и реализация API и прочее.

Однако при появлении первых ЭВМ в 1930-х годах [2] программирование использовалось для решения научных задач. Научное программирование сильно отличается от других видов программирования, при разработке научного программного обеспечения очень важно получить корректный и стабильный конечный продукт, а также четко разделить интерфейсную и научную часть. Из этого следует, что разработчик научного программного обеспечения должен быть экспертом в предментной области, а сам разработчик обычно является конечным пользователем [3], в то время как в индустриальном программировании разработчик зачастую не является конечным пользователем и от него не требуется быть экспертом в предметной области разработываемого приложения.

Также научное программирование отличается от индустриального программирования тем, что стандарты проектирования научного программного обеспечения вырабатываются существенно медленне или не вырабатываются вовсе, что приводит к отсутствию каких-либо системных подходов к разработке. Это приводит к сложностям при валидации и дальнейшей поддержке кода. Так, код, написанный эффективно и корректно, может оказаться бесполезным в том если поддержка кода оказывается затруднительной, это приводит к формированию и накоплению “best practices"при программировании численных методов. Вследствие чего стали появляться системы, которые позволяют минимизировать написание кода и снизить трудозатраты на его поддержку.

Обзор источников литературы

В основном существующие платформы используют визуальное программирование [4]. Одними из самых популярных и успешных разработок в этой области являются Simulink и LabView. Simulink позволяет моделировать вычислительные методы с помощью графических блок-диаграмм и может быть интегрирован со средой MATLAB. Также Simunlink позволяет автоматически генерировать код на языке на C для реализации вычислительного метода в

режиме реального времени. LabView используется для аналогичных задах - модерование технических систем и устройств [5]. Среда позволяет создавать виртуальные приборы с помощью графической блок-диаграммы, в которой каждый узел соответствует выполнению какой-либо функции. Представление программного кода в виде такой диаграмы делает его интуитивно понятным инженерам и позволяет осуществлять разработку системы более гибко и быстро. В составе LabView есть множество специализированных библиотек для моделирования систем из конкретных технических областей.

Существует множество других систем и языков программирования для реализации вычислительных методов, однако, каждый из них является узкоспециализированным и решает определенную задачу. Так, например, система визуального модерования FEniCS [6] используется для решения задач с использованием метода конечных элементов. Система имеет открытый исходный код, а также предоставляет удобный интерфейс для работы с системой на языках Python или C++. FEniCS предоставляет механизмы для работы с конечно-элементными расчетными сетками и функциями решения систем нелинейных уравнений, а также позволяет вводить математические модели в исходной интегрально-дифференциальной форме.

Отдельного упоминания стоит система TensorFlow. TensorFlow представляет из себя библиотеку с открытым исходным кодом, которая используется для машинного обучения. Аналогично LabView и Simulink, TensorFlow позволяет строить программные реализации численных методов. Стоит обратить внимание, что в основе TensorFlow лежит такое понятие как граф потока данных. В самом графе ребра - тензоры, представляют из себя многомерные массивы данных, а узлы - матетиматические операции над ними.

Применение ориентированных графов очень удобно для построения архитектур процессов обработки данных (как в автоматическом, так и в автоматизированном режимах). Вместе с тем многочисленные возникающие в инженерной практике задачи предполагают проведение повторяющихся в цикле операций. Самым очевидным примером является задача автоматизированного проектирования (АП). Эта задача предполагает, как правило, постановку и решение некоторой обратной задачи, которая в свою очередь, часто, решается путём многократного решения прямых задач (простым примером являются задачи минимизации некоторого функционала, которые предполагаю варьирование параметров объекта проектирования с последующим решением прямой задачи и сравнения результата с требуемым согласно заданному критерию оптимизации). Отметим, что прямые задачи (в различных областях) решаются одними методами, тогда как обратные - другими. Эти процессы могут быть очевидным образом отделены друг от друга за счет применения единого уровня абстракции, обеспечивающего определение интерпретируемых архитектур алгоритмом, реализующих методы решения как прямой, так и обратной задач. Очевидным способом реализации такого уровня абстракции стало использование ориентированных графов.

А.П.Соколов и А.Ю.Першин разработали графориентированный программный каркас для реализации сложных вычислительных методов, теоретические основы представлены в ра-

боте [1], а принципы применения графоориентированного подхода зафиксированы в патенте [7]. Заметим, что в отличии от TensorFlow, в представленном графориентированном программном каркасе узлы определяют фиксированные состояния общих данных, а ребра определяют функции преобразования данных. Для описания графовых моделей был разработан формат aDOT, который расширяет формат описания графов DOT [8], входящий в пакет утилит визуализации графов Graphviz. В aDOT были введены дополнительные атрибуты и определения, которые описывают функции-предикаты, функции-обработчики и функции перехода в целом. Подроробное описание формата aDOT приведено в [9].

В описанном в работе методе вводятся такие понятия как функции-обработчики и функции-предикаты, заметим, что функции-предикаты позволяют проверить, что на вход функции-обработчика будут поданные корректные данные. Подобная семантика функций-предикатов предполагает, что функции-предикаты и функции-обработчики должны разрабатываться одновременно в рамках одной функции-перехода. Такой подход существенно ускоряет процесс реализации вычислительного метода - функции-перехода могут разрабатываться параллельно несколькими независимыми разработчиками. Также, возможность параллелизации процесса разработки позволяет организовать модульное тестирование и документирование разрабатаываемого кода.

1 Постановка задачи

1.1 Концептуальная постановка задачи

В разработанном А.П.Соколовым и А.Ю.Першиным графоориентированном программном каркасе [1] , для описания графовых моделей был разработан формат aDOT, который является расширением формата описания графов DOT. Для визуализации графов, описанных с использоваением формата DOT, используются специальные программы визуализации. Самой популярной из них является пакет утилит Graphviz, пакет позволяет визуализировать графы описанные в формате DOT и получать изображение графа в разных форматах: PNG, SVG и т.д. Формат aDOT расширяет формат DOT с помощью дополнительных атрибутов и определений, которые описывают функции-предикаты, функции-обработчики и функции-перехода в целом. Таким образом, становится очевидно, что для построения графовых моделей с использованием формата aDOT необходим графический редактор. В представленном графоориентированном каркасе допускается наличие циклов, которые обрабатываются особым образом и требуют отдельного внимания разработчика, таким образом, разрабатываемый редактор должен предоставлять возмжность поиска циклов в загружаемой и/или создаваемой в редакторе графовой модели.

Таким образом, разрабатываемый графический редактор должен удовлетворять следующим требованиям:

- Разрабатываемый редактор должен представлять из себя web-приложение - это позволяет сделать редактор независимым от операционной системы.
- Редактор должен предоставлять возможность создавать ориентированный граф с нуля
- Возможность экспотировать созданный граф в формат aDOT
- Возможность загрузить граф из формата aDOT
- Предоставлять возможность поиска циклов с последующей их подсветкой в загружаемой и/или создаваемой в редакторе графовой модели

2 Программная реализация

1 Создание примитивов на странице

Как было ранее сказано, редактор графов является web-приложением написанным на языке JavaScript. Для создания графа необходима возможность создавать простейшие примитивы: окружности, линии и прочее. JavaScript предоставляет несколько вариантов для решения этой задачи:

- Использование HTML – элемента canvas
- Работа с векторной графикой svg

Использование canvas не подходит поскольку после создания примитива невозможно получить его свойства, а следовательно и отредактировать их. Таким образом, необходимо работать с векторной графикой svg, использование svg позволяет создавать примитивы, которые создаются как HTML теги с набором свойств, что позволяет редактировать или удалять созданные примитивы. Однако работать с svg без использования сторонних библиотек достаточно затруднительно, программный код сильно увеличивается и разработка функций для отрисовки примитивов занимает существенно больше времени. Для решения этой задачи подходит библиотека d3.js. Библиотека предоставляет набор инструментов для визуализации данных на странице, который состоит из нескольких дестяков небольших модулей, каждый из которых решает свою задачу. В программном коде возможности d3 используются для создания svg примитивов на странице, например создание вершины (листинг 2.1).

```
1   svg
2     .append("circle")
3     .attr("cx", x)
4     .attr("cy", y)
5     .attr("r", radius)
6     .attr("stroke-width", borderWidth)
7     .attr("id", id)
8     .attr("fill", "#FFFFFF")
9     .attr("stroke", "#000000")
10    .attr("class", "vertex");
```

Листинг 2.1. Пример создание вершины с использование библиотеки d3

2 Хранение информации о графовой модели

Для того, чтобы иметь возможность гибко редактировать графовую модель, а также иметь возможность экспортировать граф в формат aDOT, необходимо хранить множество свойств графа. Так, при удалении вершины, помимо удаления примитивов со страницы, необходимо удалить всю информацию о вершине и связанных с ней ребрах, в противном случае при

сохранении графа в формате aDOT в файле будет содержаться информация об уже удаленных вершинах или ребрах.

Все свойства графа хранятся в оперативной памяти. Для хранения большей части свойств в программном коде используются объекты. Объект - это мощная структура данных в javascript - используется для хранения коллекций разных значений. Рассмотрим на примере как храниться информаци о вершинах. В рассматриваемом примере (листинг 2.2) представлены хранимые в объекте свойства одной вершины из которой выходит одно ребро.

```
{
    "edges": [
        {
            "direction": "from",
            "function": "f1",
            "predicate": "p1",
            "label": "<p1, f1>",
            "metadata": {
                "edgeID": "edge1",
                "pathID": "edge1_path",
                "labelID": "edge1_label",
            }
            "type": "straight",
            "value": "vertex2",
        },
    ],
    "metadata": {
        "label": "s1",
        "label_metadata": {
            "pathID": "vertex1_path",
            "labelID": "vertex1_label",
        },
        "position": {
            "x": 227.609375,
            "y": 211,
        },
    }
}
```

Листинг 2.2. Пример хранение вершины в оперативной памяти

Аналогичным образом с помощью объектов храниться информация о предикатах и функциях, которые пользователь вводит в процессе создания графовой модели. Для хранения более простых блоков данных используются массивы, например, хранение позиций уже созданных вершин, или хеш-таблицы, например, хранение уже созданных меток вершин.

3 Поиск циклов в графовой модели

Разработанный редактор позволяет найти циклы в создаваемой или загружаемой графовой модели. Для поиска циклов в графе реализован алгоритм в основе которого лежит DFS(Depth-first search) - поиск в глубину. Алгоритм является рекурсивным: для рассматриваемой вершины необходимо перебрать все ребра, которые выходят из нее, если ребро приходит в нерасмотренную ранее вершину, то алгоритм запускается от этой вершины. Возврат происходит в том случае, если все выходящие из рассматриваемой вершины ребра идут в уже расмотренные ранее вершины. В результате работы алгоритма будут рассмотрены все вершины графа.

Стоит обратить внимание, что в интернете представлено множество различных реализаций поиска циклов в графе, однако почти все предложенные реализации выполняют проверку графа на ацикличность, то есть нахождение любого цикла в графе с последующим утверждением, что граф цикличен или ацикличен в противном случае. В то время как в разработанном редакторе необходимо найти все циклы в графе. В основе реализованного алгоритма также лежит DFS, а для нахождения всех циклов в графе поиск в глубину запускается от всех нерасмотренных ранее вершин. Алгоритм состоит из двух функций: функция для поиска в глубину и соответственно вызывающая функция. Далее на рисунках (1) и (2) приведены блок-схемы этих функций.

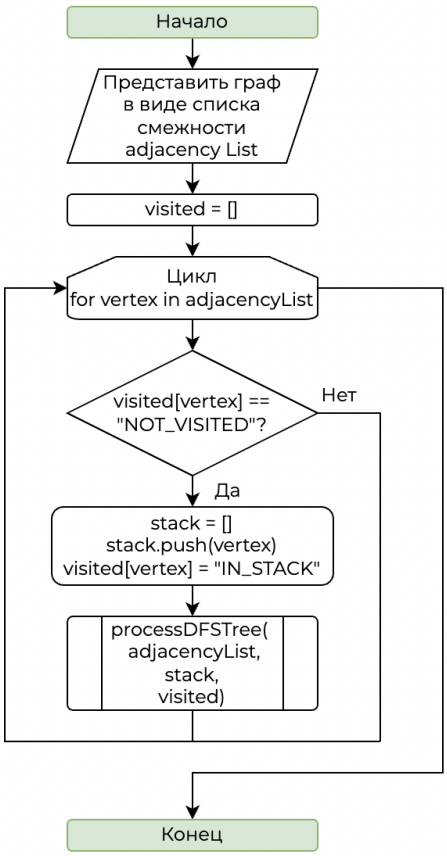

Рисунок 1. Главная функция алгоритма поиска циклов в графе

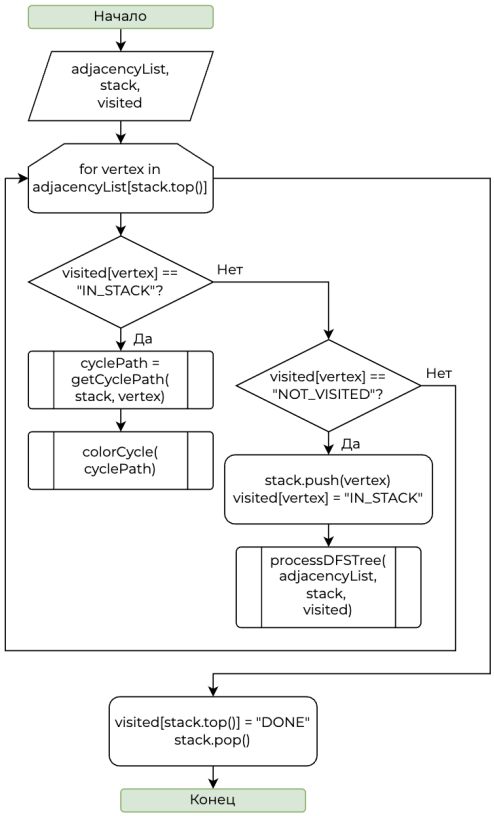

Рисунок 2. Функция выполняющая обход в глубину

Разработанный алгоритм позволяет находить все циклы в графе и выполнять из подсветку. В следующем разделе приведены примеры работы алгоритма на разных графовых моделях.

3 Тестирование и отладка

Реализованный алгоритм поиска циклов в графе позволяет находить все циклы в загружаемой или создаваемой графовой модели и выполнять их подсветку. В качестве примеров рассмотрим несколько графовых моделей: на рисунке (3) приведен граф, который не содержит ни одного цикла, соответственно после выполнения поиска циклов не будет найдено ни одного цикла, поэтому ни вершины ни ребра подсвечены не будут.

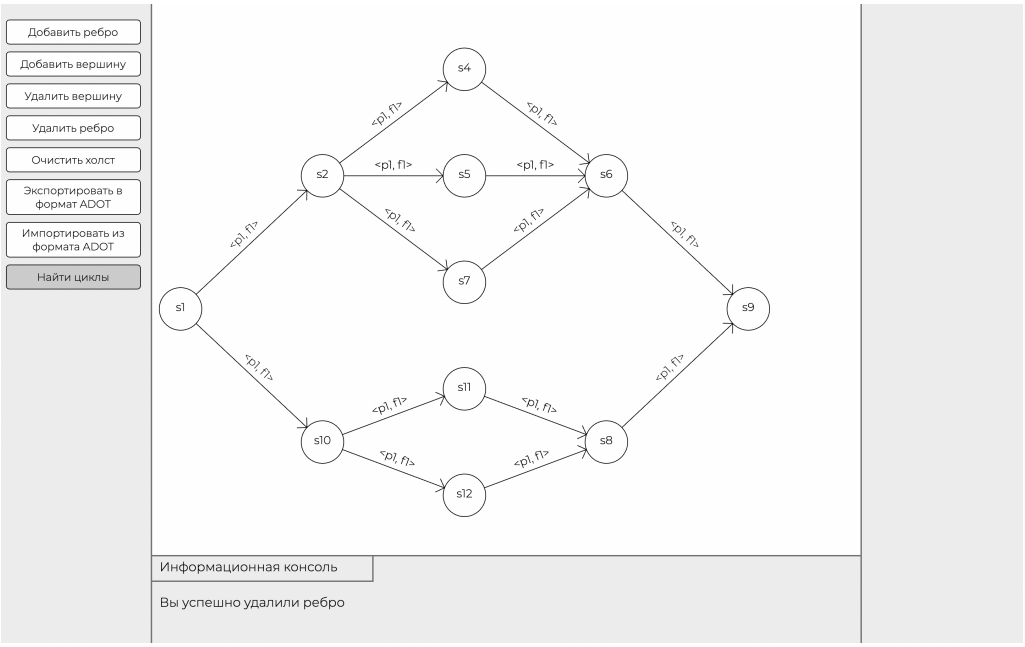

Рисунок 3. Граф без циклов

На рисунке (4) в графе присутствует только один цикл, после выполнения поиска циклов данный цикл будет найден и подсвечен.

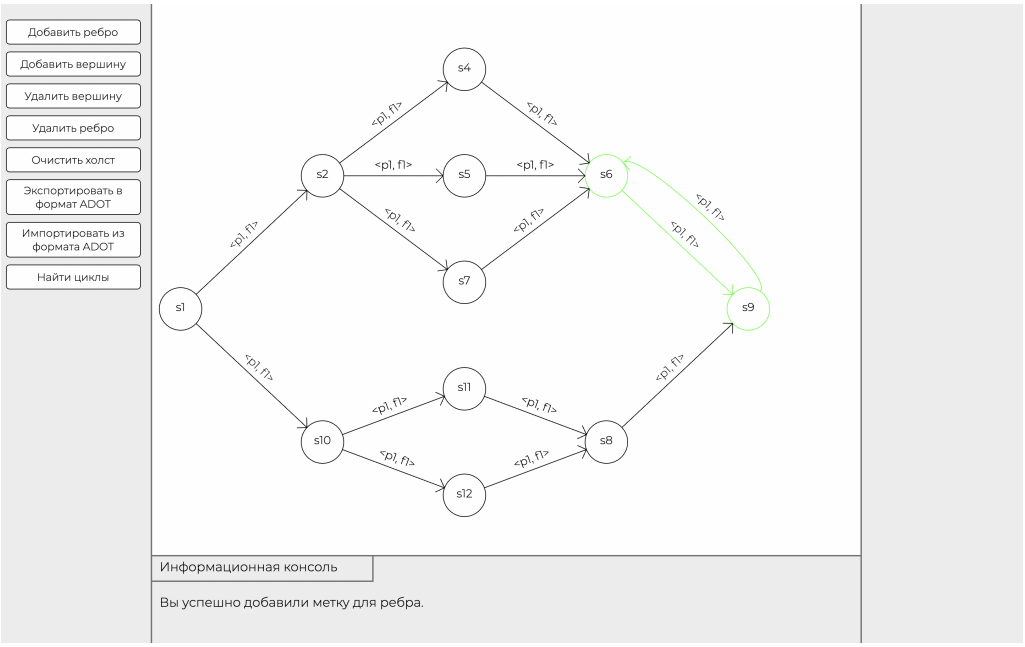

Рисунок 4. Граф с одним простым циклом

В приведенном на рисунке (5) графе присутствует множество циклов, в результате поиска циклов абсолютно все циклы были найдены и подсвечены.

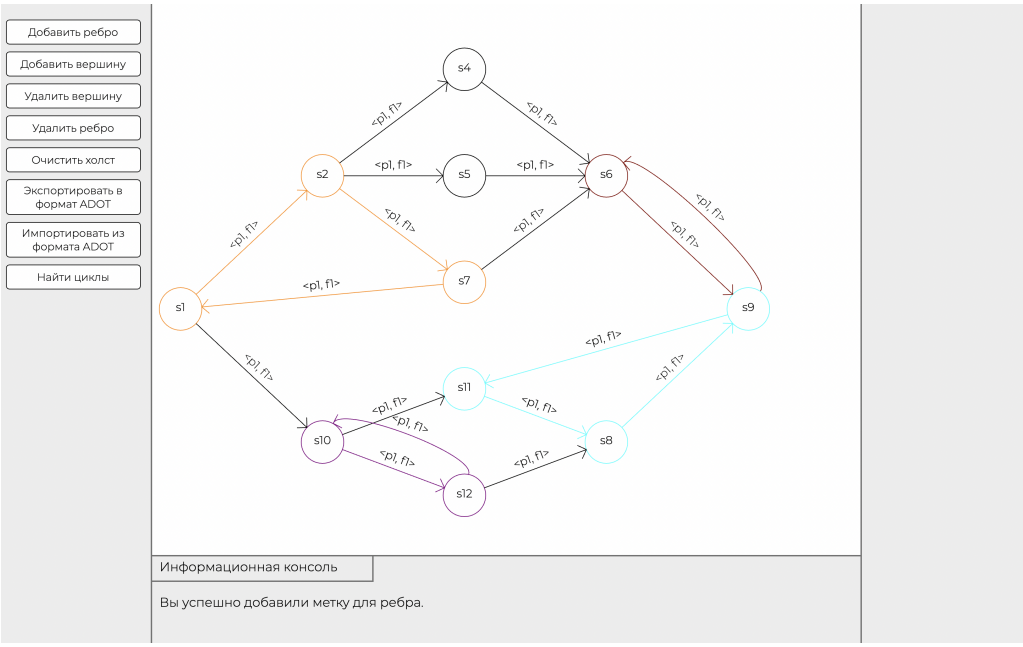

Рисунок 5. Граф с множеством циклов

4 Анализ результатов

В примерах продемонстрировано, что разработанный редактор позволят выполнить поиск циклов в загружаемой или создаваемой в редакторе графовой модели. Для решения задачи поиска циклов использовался алгоритм поиска в глубину (DFS). Рассмотрим скорость работы алгоритма на примере. Граф на котором будет запускаться поиск циклов приведен на рисунке (6).

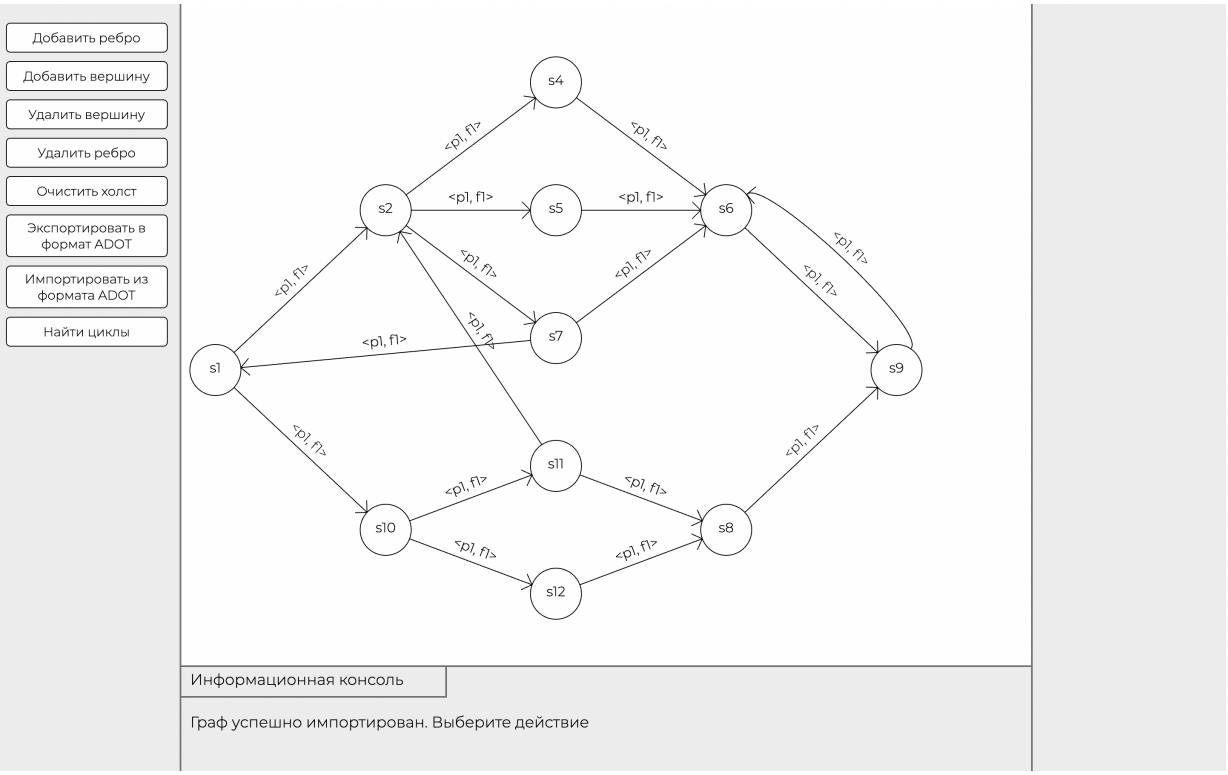

Рисунок 6. Граф для анализа скорости работы алгоритма поиска циклов

В результате многократного запуска поиска циклов в рассматриваемом графе были получены следующие результаты:

- 0.254150390625 ms
- 0.206787109375 ms
- 0.260986328125 ms
- 0.236083984375 ms
- 0.3359375 ms

Алгоритм работает достаточно быстро, однако перед выполнением поиска в глубину необходимо представить граф в виде списка смежности. Вычисление списка смежности происходит при каждом запуске поиска циклов, что не является оптимальными решением. Если графовая модель не изменяется, то повторный запуск поиска циклов будет вычислять один и тот же список смежности несколько раз. Таким образом, в качестве оптимизации алгоритма список смежности можно получать "на лету постепенно формировать, в случае создания гра-

фовой модели в редакторе, а в случае загрузки графовой модели получить список смежности сразу после успешной визуализации графа.

ЗАКЛЮЧЕНИЕ

В результате проведенной работы был реализован редактор графов, который является достаточно полезным инструментом, дополняющим предложенный А.П.Соколовым и А.Ю.Першиными графоориетированный программный каркас для реализации сложных вычислительных методов. Разработанный редактор позволяет создавать ориентированный граф с нуля, а также выполнять поиск циклов в созданной или загруженный из формата aDOT графовой модели. Возможность поиска циклов позволяет разработчику вычислительного метода, вместого ручного поиска циклов на графовой модели, одним действием найти все циклы в графе, а реализованная подсветка найденных циклов позволит визуально отличить один цикл от другого.

Список использованных источников

1 Соколов А.П., Першин А.Ю. Графоориентированный программный каркас для реализации сложных вычислительных методов. 2019.

2 Timothy Williamson. History of computers: A brief timeline. 2021.

3 J. E. Hannay C. MacLeod J. S. e. a. How do scientists develop and use scientif-ic software? 2009.

4 Robert van Liere. CSE. A Modular Architecture for Computational Steering. 2015.

5 А.В. Коргин М.В. Емельянов В.А. Ермаков. Применение LabView для решения задач сбора и обработки данных измерений при разработке систем мониторинга несущих конструкций. 2013.

6 Mortensen Mikael Langtangen Hans Petter Wells Garth N. A FEniCS-Based Programming Framework for Modeling Turbulent Flow by the Reynolds-Averaged Navier-Stokes Equations. 2011.

7 Соколов А.П., Першин А.Ю. Патент на изобретение RU 2681408. Способ и система графо-ориентированного создания масштабируемых и сопровождаемых программных реализаций сложных вычислительных методов. 2019.

8 Stephen C. North Eleftherios Koutsofios. Drawing Graphs With Dot. 1999.

9 Соколов А.П. Описание формата данных aDOT (advanced DOT) [Электронный ресурс]. Облачный сервис SA2 Systems. [Офиц. сайт]. 2020.