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

ФАКУЛЬТЕТ *Робототехника и комплексная автоматизация*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ*

## *НА ТЕМУ*

«Разработка web-приложения, обеспечивающего построение графовых моделей алгоритмов обработки данных»

Студент РК6-82Б (Группа) _____ (подпись, дата) Н.В. Журавлев (инициалы и фамилия)

Руководитель ВКР _____ (подпись, дата) А.П. Соколов (инициалы и фамилия)

Нормоконтролер _____ (подпись, дата) С.В. Грошев (инициалы и фамилия)

*Москва, 2023 г.*

## РЕФЕРАТ

Выпускная квалификационная работа: 44 с., 15 рис., 10 источн.

Работа посвящена разработке визуализатора графовых моделей, с некоторыми особенностями. А именно, граф должен описывать объект проектирования, который имеет какие-либо параметры, которые должны изменяться в графе. Узлами графа является определенные значения параметров изучаемого объекта. В ребрах должен исполняться код на языке Python, предварительно занесённый туда пользователем. Этот код должен изменять какие-либо параметры объекта проектирования. При разработке для визуализации графа был использован алгоритм dot, а для обхода графа был создан собственный.

Тип работы: выпускная квалификационная работа.

Тема работы: «Разработка web-приложения, обеспечивающего построение графовых моделей алгоритмов обработки данных».

Объект исследования: архитектура процессов обработки данных.

Основная задача, на решение которой направлена работа: создание визуализатора графовых моделей, который должен описывать объект проектирования, который имеет какие-либо параметры, изменяемые в процессе обхода графа.

Цели работы: создать основу для построения/сохранения/накопления графовый моделей, описывающих тот или иной процесс обработки данных

В результате выполнения работы:

1. Предложено создание визуализатора графовых моделей;
2. Создан алгоритм обхода ориентированного графа;
3. Разработано алгоритмы визуализации и обхода графа. Возможность чтения и вывод из файла формата aDot;

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

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. 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 невязки при варьировании параметров объекта проектирования. Каждая итерация сопряжена с решением прямой задачи и сравнением численного результата с требуемым (например, известным из эксперимента). Функционал строят согласно заданному критерию оптимизации.

Необходимо отметить, 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 алгоритмов, реализующих методы решения как прямой, так и обратной задач. Очевидным способом реализации такого уровня абстракции стало использование ориентированных графов.

Целью работы является обязательно создать основу для построения, сохранения, накопления графовый моделей, описывающих тот или иной процесс обработки данных.

Достаточно сложным найти материалы по технологиям разработки специализированного наукоемкого ПО (в том числе коммерческого). Примерами такого может быть: различное инженерное ПО, CAD-, CAE-системы и ПО математического моделирования.

Особенностью разработка наукоемкого ПО является сложность реализуемых алгоритмов и нужда в реализации сложных вычислительных методов. При создании такого ПО, проблемы отладки разрабатываемой

программы встают все более остро, такие как: множество частей исходного кода, потенциально содержащих в них ошибки, что ведет к существенно большим трудозатратам на отладку для получения конечного результата. Увеличение числа разработчиков еще больше усугубляет первую проблему, так как приводит к большей несогласованности при внесении изменений в исходный код общей системы.

Частичное решение этой проблемы предоставляют системы контроля версий (Subversion, GIT и др.), но и они не позволяют обеспечить логическую согласованность вносимых изменений. Каждый разработчик не может заранее знать всю архитектуру большой системы, но должен в ней вести работы, что небезопасно для целостности исходного кода – возможно нарушение принципов построения.

Для ПО рассматриваемого класса характерна увеличивающаяся ресурсоемкость, что требует наличия многопроцессорных высокопроизводительных компьютеров. Такие требования могут вызвать необходимость переработки ранее написанного кода, включая разработку параллельных версий уже созданных последовательных программ.

Широкий спектр приложений, потенциальная ресурсоемкость актуальных прикладных вычислительных задач, высокие требования к разработчикам - из всего этого получается высокая сложность разработки программных средств из рассматриваемого класса и появляется необходимость использования специализированных технологий разработки.

Известным подходом, в том числе применяемым для разработки ПО указанного класса, является организация комплексных вычислений или описание алгоритма решения вычислительной задачи в форме ориентированного графа или «графовой модели» алгоритма (содержащего или не содержащего циклы). Далее с использованием интерпретатора осуществляется обход графовой модели.

Однако, как правило системы из указанного класса применяются далеко не только для организации вычислений, а являются системами решения задач

организации процессов обработки данных и используются в различных прикладных областях (например, описание бизнес-процессов, разработка систем реального времени и др.), что тоже влияет на актуальность данной разработки. Но в данной работе указанный подход рассматривается только в контексте разработки вычислительных подсистем.

Основной технологией, реализованной и примененной при разработке, описываемой в текущей работе вычислительной подсистемы, стал графоориентированный подход или графоориентированная программная инженерия. Особенностями подхода являются существенные различия в назначении узлов и ребер, а также использование ориентированных графов общего вида, допускающих наличие циклов, что, как правило, является ограничением многих известных систем, использующих ориентированные ациклические графы.

Применение подхода позволило бы систематизировать процесс разработки программных реализаций сложных вычислительных методов и обеспечило бы возможность формирования вычислительных библиотек группой разработчиков независимо друг от друга.

Фактическое отсутствие в открытой печати информации об отечественных системах рассматриваемого класса обусловлено, предположительно, закрытостью большинства из них и применением, как правило, в коммерческих целях.

Необходимым условием, определяющим принципиальную возможность создания ПО рассматриваемого типа, является участие специалистов, обладающих одновременно компетенциями как в области математического моделирования, так и в области программной инженерии. До 2006 г. специалистов по программной инженерии высшие учебные заведения России не готовили вовсе. Как правило, специалисты в области математического моделирования не обладают достаточными навыками в области программной инженерии и наоборот.

Помимо специализированных технологий принципиально важным при создании сложных программных комплексов является непосредственное участие руководителя проекта в отборе и подготовке кадров. Руководитель должен быть признанным лидером коллектива, что определяется его профессиональными компетенциями.

Все перечисленные проблемы определили очевидное отставание и неконкурентоспособность России в рассматриваемой отрасли в сравнении с существующими в мире программными решениями [1].

## 1. Описание вычислительной подсистемы

Программное обеспечение, предназначенное для решения вычислительных задач из одного класса, будем называть решателем. Решатель является программной реализацией некоторого вычислительного метода или, в более общем случае, сложного вычислительного метода.

Вычислительный метод, предполагающий использование одного или нескольких других вычислительных методов, назовем сложным (далее сложный вычислительный метод (СВМ)).

Используя введенную терминологию, вычислительная подсистема, помимо прочего, должна объединять в своем составе множество программных реализаций сложных вычислительных методов, обеспечивающих решение вычислительных задач рассматриваемых классов.

Требования к вычислительной подсистеме в целом индуцируются требованиями к ее составным частям, то есть к решателям. В свою очередь требования к решателям определяются особенностями классов вычислительных задач, для решения которых они предназначены.

Алгоритм создания графоориентированной реализации СВМ сводится к выполнению следующих процедур:

1. Определить перечень входных данных и представить их в виде файла, который далее будет загружен в вычислительную подсистему.
2. Определить множество состояний данных, каждому из которых должен соответствовать момент успешного завершения очередной процедуры

обработки данных, представлено на рис. 1. Среди состояний выделить начальное и конечное. Начальное состояние данных соответствует моменту, когда входные данные, определенные в первом пункте, были загружены в оперативную память.

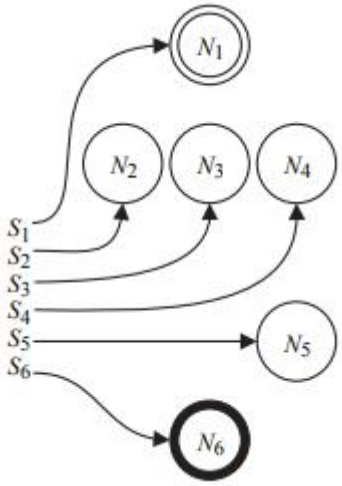


Рисунок 1. Связывание множество состояний и узлов графа. [1]

3. Определить множество функций-обработчиков данных, согласно реализуемому СВМ.
4. Каждому состоянию сопоставить узлы будущей графовой модели.
5. Связать узлы ребрами, связывающими состояния данных.
6. Каждому ребру поставить в соответствие функцию перехода, определяемую парой f - функция-обработчик, p – функция-предикат.
7. Определять или дополнять множество функций-предикатов и множество функций-селекторов, также связывая их с соответствующими ребрами или узлами согласно их назначению.

Перечисленные пункты алгоритма следует выполнять с использованием языка определения графовых моделей aDOT. В результате проведенных процедур будет построена графовая модель необходимого СВМ в формате aDOT. Внешний вид графа представлен на рис. 2.

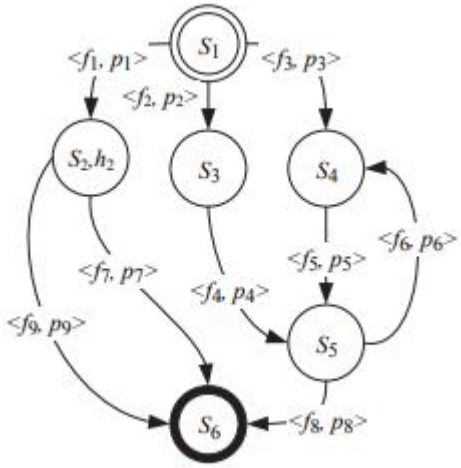


Рисунок 2. Итоговый вид необходимого СВМ [1]

Необходимо отметить, что функции-обработчики, функции-предикаты и функции-селекторы необходимо группировать по назначению и представлять в виде программных библиотек.

В рамках представляемой работы вычислительной подсистемы непосредственная разработка функций-обработчиков, функций-предикатов и функций-селекторов осуществляется на языке программирования Python.

Интерпретация (обход и выполнение) графовой модели, в свою очередь, возможна только при предварительной реализации всех необходимых функций-обработчиков, функций-предикатов и функций-селекторов, их размещении в библиотеках согласно определению графовой модели и организации к ним доступа интерпретатору.

Представляемый подход позволяет создавать в определенном смысле «универсальные» графовые модели, реализующие одновременно различные СВМ. Другими словами, сопоставляя с элементами графовой модели различные комбинации функций-обработчиков, функций-предикатов и функций-селекторов можно формировать реконфигурируемые графовые модели с единой топологией.

Параллельные ветви графовой модели по умолчанию имеют одинаковый приоритет и могут выполняться параллельно. Однако существуют ситуации, когда параллельная обработка невозможна или нецелесообразна. Например, при отладке разработанного графоориентированного решателя удобен

последовательный обход графа (рис. 3, а). В то же время, параллельная обработка зависит от наличия доступных аппаратных ресурсов, а также от программной поддержки на уровне интерпретатора графовых моделей. Более того, бывают ситуации, когда требуется обеспечить не распараллеливание, а ветвление (рис. 3, б).

Таким образом, в процессе обхода графовой модели, «дойдя» до некоторого узла, из которого «выходят» несколько ребер, на уровне интерпретатора должна быть возможность принять решение о методе дальнейшего обхода. В рамках представляемой реализации графоориентированного подхода принятие такого решения осуществляется: а) определением стратегии распараллеливания или ветвления в узле; б) связыванием с узлом функции-селектора.

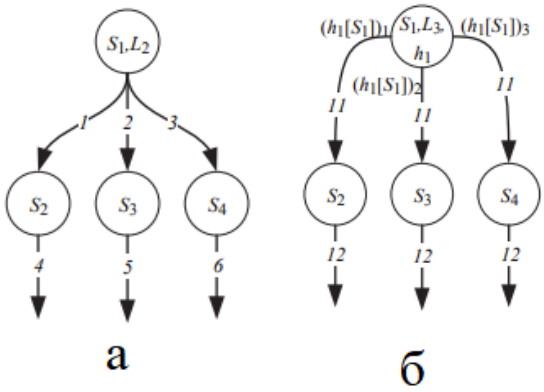


Рисунок 3. а) Последовательный обход б) Ветвление при обходе [1]

Для визуализации графов, описанных с использованием формата DOT, используются специальные программы визуализации. Формат aDOT расширяет формат DOT с помощью дополнительных атрибутов и определений, которые описывают функции-предикаты, функции-обработчики и функции-перехода в целом. Из этого вытекает, что для построения графовых моделей с использованием формата aDOT необходим графический редактор.

### 1.1. Обзор алгоритмов визуализации графов

Чтобы выбрать нужный для данного случая алгоритм необходимо выделить свойства графа олицетворяющий СВМ, по котором будет произведён отбор.

1. Граф является ориентированным.

2. Конец графа должен находится в противоположной стороне от начала.
3. Максимально допустимый граф является среднего размера.
4. В графе могут иметься циклы.

Для визуализации графов для рассмотрения выбрано два способа визуализации графов - через силовые алгоритмы визуализации графов и через алгоритмы послойного рисования графа [2].

Силовые алгоритмы визуализации графов используются для создания графического представления графа, который показывает связи между вершинами. Эти алгоритмы основаны на физических законах, которые определяют движение объектов в пространстве.

Далее представлены общие шаги для рисования графа через силовые алгоритмы визуализации графов.

1. Определение вершины и ребра графа.
2. Инициализация каждой вершины случайными координатами в пространстве.
3. Определение силы, действующие на вершины графа. Силы могут быть притягивающими или отталкивающими.
4. Расчёт силы, действующие на каждую вершину, и перемещение ее в соответствии с этими силами.
5. Повторение шаги 3 и 4 до тех пор, пока вершины не расположатся в необходимом порядке.

Послойное рисование графа - это метод визуализации графа, при котором узлы располагаются на разных слоях, а связи между ними проходят только между соседними слоями.

Способ послойного рисования графа можно описать следующим образом.

1. Определение слоёв графа. Для этого можно использовать любой алгоритм сортировки, который определяет порядок узлов в графе таким образом, чтобы все связи шли от узлов с более низким индексом к узлам

с более высоким индексом. Таким образом, каждый узел будет располагаться на слое с номером, соответствующим его индексу.

2. Размещение узлов на слоях. Узлы на каждом слое могут быть расположены в любом порядке, но часто используется выравнивание по центру или по левому краю слоя.
3. Рисование связи между узлами. Связи должны проходить только между соседними слоями, то есть связи между узлами на одном слое не должны быть нарисованы, хотя существуют алгоритмы, позволяющие это сделать. Для рисования связей можно использовать прямые линии или кривые, в зависимости от визуальных предпочтений.
4. Добавление дополнительных элементов. В зависимости от конкретной задачи, к графу можно добавить дополнительные элементы, такие как подписи узлов или связей, стрелки направления связей и тому подобное.
5. Оптимизация расположения элементов. После первоначального размещения узлов и связей можно провести оптимизацию расположения элементов, чтобы минимизировать пересечения связей и узлов и обеспечить лучшую визуализацию графа.

Для размещения графов известны силовые алгоритмы следующих типов:

1. Force-Directed - вершины представляются как заряженные частицы, которые отталкивают друг друга с помощью физической силы, а рёбра — как упругие струны, которые стягивают смежные вершины
2. Multidimensional scaling - то класс алгоритмов визуализации данных, который используется для представления многомерных данных в двумерном или трехмерном пространстве. Он основан на принципе сохранения расстояний между точками в исходном пространстве в новом пространстве.
3. Energy-Based - в этом подходе пытаются описать потенциальную энергию системы и найти положение вершин, которое будет соответствовать минимуму.

Рассмотрим самые часто упоминаемые алгоритмы, а именно: Fruchterman-Reingold, Алгоритм Идеса, Kamada Kawaii, Force Atlas 2, OpenOrd, Гамма – алгоритм.

В алгоритме Fruchterman-Reingold используется пружинная физическая модель, в которой каждый узел рассматривается как заряд, а ребра - как пружины. Силы могут действовать только на вершины, вес пружин при этом не учитывается. Целью алгоритма является минимизация энергии системы, которая определяется как сумма кинетической и потенциальной энергии.

Далее представлены основные шаги алгоритма.

1. Вершины графа помещаются в случайные координаты.
2. Рассчитываются силы, действующие на вершины.
3. Происходит перемещение вершин, проверяется выход за границу экрана.
4. Повторяются шаги 2-3.

Алгоритм Fruchterman-Reingold продолжает выполняться до тех пор, пока система не достигнет минимальной энергии или пока не будет достигнуто максимальное количество итераций. В результате работы алгоритма получается оптимальное размещение узлов на плоскости, которое минимизирует пересечения ребер и узлов и обеспечивает лучшую визуализацию графа.

Алгоритм начинается с случайного размещения узлов на плоскости. Затем для каждого узла вычисляется сила отталкивания от всех остальных узлов, которая определяется по закону Кулона. Также для каждой пары связанных узлов вычисляется сила притяжения, которая определяется по закону Гука. Эти две силы влияют на перемещение каждого узла в следующем шаге.

На каждом шаге алгоритма происходит перемещение каждого узла в направлении суммарной силы, действующей на него.

Алгоритм не имеет механизма остановки, а значит пользователь сам решает, когда закончить работу алгоритма. Обычно при реализации ограничивают количество итераций. Для достижения хорошего результата обычно 100 итераций достаточно [3].

Данные алгоритм не подходит, так как он не предусмотрен для ориентированных графов.

Алгоритм Kamada Kawaii похож на алгоритм Fruchterman-Reingold, но выбирается вершина, на которую действует максимальная сила, затем остальные вершины фиксируются, энергия системы минимизируется.

Далее представлены основные шаги алгоритма.

1. Рассчитываются расстояния по графу между всеми вершинами.
2. Вершины графа помещаются в случайные координаты.
3. Выбирается вершина, на которую действует максимальная сила.
4. Остальные вершины фиксируются, энергия системы минимизируется.
5. Повторяются шаги 3-4.

Алгоритм не имеет механизма остановки, а значит пользователь сам решает, когда закончить работу алгоритма. При реализации ограничивают количество итераций.

Данный метод не подходит, так как имеет самую высокое время работы из рассматриваемых - O($V^3$) [4].

Force Atlas 2 представляет граф в виде металлических колец, связанных между собой пружинами. Деформированные пружины приводят систему в движение, она колеблется и в конце концов принимает устойчивое положение.

Далее представлены основные шаги алгоритма.

1. Алгоритм начинается работу с случайным расположением вершин на плоскости.
2. Определение локальных сил. Для каждой вершины вычисляются силы, действующие на нее от ее соседей. Каждая вершина притягивается к ближайшим соседям и отталкивается от более далеких вершин.
3. Расчет глобальных сил. Дополнительные силы применяются к вершинам, чтобы удерживать их внутри области рисунка и предотвратить их пересечение.

4. В процессе итерирования вычисляется общая сила, действующая на каждую вершину, и изменяются ее координаты.
5. Алгоритм продолжается до тех пор, пока сила, действующая на каждую вершину, не станет достаточно малой.

Основная его задача - визуализация графов, в которых имеются подмножества с высокой степенью взаимодействия, таким свойством целевой граф не обладает [5].

OpenOrd - это алгоритм, специально предназначенный для очень больших сетей, который работает на очень высокой скорости при средней степени точности. Это хороший компромисс для больших сетей, но часто он нежелателен для небольших графов где потеря точности может быть значительной по сравнению с другими подходами к компоновке. Все вершины изначально помещаются в начало координат, а затем проводятся итерации оптимизации. Основные шаги алгоритма:

1. Алгоритм начинается с случайного расположения вершин на плоскости.
2. Определение локальных сил. Для каждой вершины вычисляются силы, действующие на нее от ее соседей. Каждая вершина притягивается к ближайшим соседям и отталкивается от более далеких вершин. Эти силы используются для определения веса ребер между вершинами.
3. Расчет глобальных сил. Дополнительные силы применяются к вершинам, чтобы удерживать их внутри области рисунка и предотвратить их пересечение.
4. Итерация. В процессе итерации вычисляется общая сила, действующая на каждую вершину, и изменяются ее координаты.
5. Алгоритм продолжается до тех пор, пока сила, действующая на каждую вершину, не станет достаточно малой.

Предназначен для визуализации больших графов, следовательно, не подходит для целевого графа, так как он является графом маленького или среднего размера [6].

Гамма - алгоритм основная идея которого заключается в том, что выделяются сегменты, затем в минимальном выделяется цепь, которая укладывается в любую грань, вмещающую данный сегмент.

1. Выбирается простой цикл в исходном графе и изображается на плоскости.
2. Пункты 3-7 повторяется до тех пор, пока граф не будет уложен или пока не будет получено, что граф не планарен.
3. Строится множество сегментов.
4. Для каждого сегмента вычисляется величина |Г(S)|.
5. Если существует i: $|\Gamma(S_i)| = 0$, то граф не планарен, алгоритм завершает работу.
6. Выбирается сегмент с минимальным числом $|\Gamma(S_i)|$ в этом сегменте выбирается цепь между двумя контактными вершинами.
7. Эта цепь укладывается в любую грань, вмещающую данный сегмент.
8. Либо получена плоская укладка графа, либо граф оказался не планарен.

Не подходит, так как в целевом графе может не оказаться циклов [7].

Алгоритм Идеса (Magnetic-Spring Algorithm) рёбра назначаются магнитной пружиной, и затем при воздействии магнитных полей вершины перемешаются [8].

Далее представлены основные шаги алгоритма.

1. Алгоритм начинается с случайного расположения вершин на плоскости.
2. Расчет сил. Для каждой вершины вычисляются магнитные силы, направленные к соседним вершинам, а также пружинные силы, направленные к вершинам, соединенным ребром.
3. Итерация. В процессе итерации вычисляется общая сила, действующая на каждую вершину, и изменяются ее координаты.

4. Условия остановки. Алгоритм продолжается до тех пор, пока не будет достигнута определенная точность или пока не будет достигнуто максимальное число итераций.
5. Отображение. После завершения работы алгоритма вершины графа могут быть отображены на плоскости с использованием их финальных координат.

За счёт возможности направить силу притяжения в определённом направлении, что улучшит удобочитаемость, является хорошим выбором.

Для любого из способа существуют графы, при визуализации которых может появиться неудовлетворительный вид. Однако в следствии того, что послойное рисование графа представляет более привычный вид ориентированного графа. Исходя из этого для решения задачи визуализации графа, бы выбран алгоритм dot.

Данный алгоритм состоит из 4 этапов - rank, ordering, position, make-splines.

В первом этапе каждому узлу необходимо определить ранг для всех узлов. Определение ранга происходит в результате "естественного" обхода. Наличие циклов не позволяет корректно пройтись по графу, поэтому необходимо разорвать циклы. Разрываются они через использование алгоритма DFS, после для каждого узла уже рассчитывается ранг.

После присвоения ранга, ребра между узлами, которые отличаются на более чем на один ранг заменяются цепочками ребер единичной длины между временными или «виртуальными» узлами. Виртуальные узлы размещаются на промежуточных рангах, превращая исходный граф в граф, ребра которого соединяют только узлы на соседних рангах. Порядок вершин в рангах определяет количество пересечений ребер, поэтому хорошим порядком является тот, который имеет наименьшим количеством пересечений.

Третий проход является этап корректировки. Он нужен для того, чтобы избежать плохих расположений узлов графа. Координаты X и Y вычисляются в два отдельных шага. На первом этапе всем узлам (включая виртуальные узлы) присваиваются координаты X в соответствии с уже определенным порядком

внутри рангов. На втором этапе назначаются координаты Y, присваивая одинаковое значение узлам одного ранга.

После этого в последнем этапе определяются и рисуются ребра, которые будут представлены в виде сплайна. Алгоритм попытается найти самую гладкую кривую между двумя точками, которая избегает «препятствий» в виде других узлов или сплайнов. Затем разделит алгоритм маршрутизации сплайнов на верхнюю половину и нижнюю половину. Верхняя половина вычисляет многоугольную область макета, где может быть нарисован сплайн. Она вызывает нижнюю половину для вычисления наилучшего сплайна в области [9].

## 1.2.Описание алгоритма обхода графа

Особенностями задачи, решаемой в рамках, данной работы является наличие селекторов и тот факт, что обход по графу не всегда осуществляется по всем вершинам. Из-за наличия данных особенностей было принято решение о создании собственного алгоритма.

Рассмотрим некоторый узел $S_i$ орграфа G, из которого выходит множество рёбер $E_i = \{e_{ij}\}_1^n, n > 0$. Для построения алгоритма обхода необходимо определить правила перехода от узла $S_i$ к другим узлам.

1. Если n = 1, то переход безусловный, т.е. предполагает вызов связанной с $e_{ij}$ функции перехода $F_{ij}$, где j соответствует номеру следующего узла $S_j$.
2. Если n > 1, то переход следует осуществлять по всем рёбрам, которые определяются с помощью функции-селектора $h_i$ (организует ветвление, рис. 4), сопоставленной с узлом $S_i$, результат выполнения которой определяет подмножество $\hat{E}_i = \{a_{ik}: a_{i,k}\big(h_i(D_i)\big) = 1, D_i \multimap S_i\}$, где $S_i$ - состояние данных СВМ, сопоставленное с одноимённым узлом $S_i$, $D_i$- данные в состоянии $S_{i,i}(r)$ - операция проекции объекта r на i-ю координату. Порядок выбора рёбер из $\hat{E}_i$ для осуществления перехода не важен.

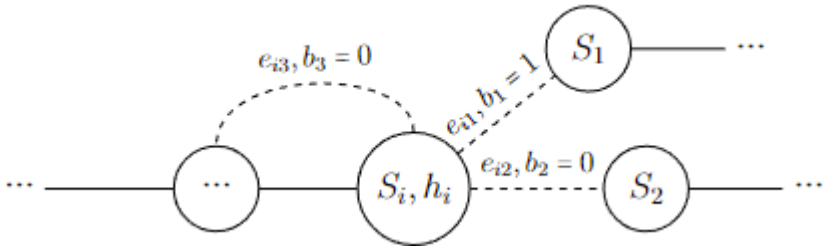


Рисунок 4. Организация ветвления с помощью функции-селектора $\boldsymbol{h_i}$ из состояния $\boldsymbol{S_i}$

Реализация $\hat{E}_i$ для каждого i возможна путём формирования структуры данных стек. Заполнение соответствующего стека элементами может быть осуществлено в произвольном порядке. При этом должны быть выполнены все функции перехода $F_{ij}$, связанные с соответствующими рёбрами, в произвольном порядке..

Рассмотрим работу алгоритма на примере орграфа, представленного на рис. 5. Начальная вершина называется Start, а конечная End. Селекторы в данном примере отсутствуют.

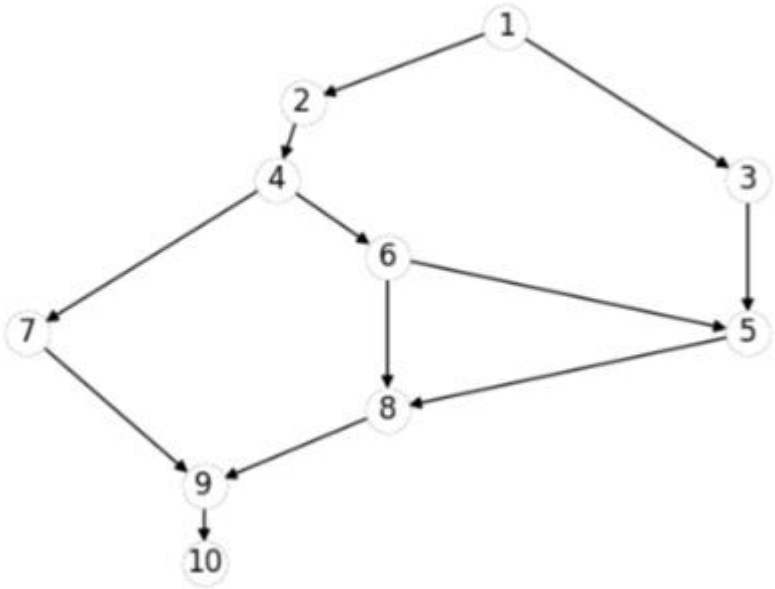


Рисунок 5. Граф для рассмотрения алгоритма обхода

Рассмотрим алгоритм перехода из узла Start. Так как из узла Start выходит два ребра, то выбираем случайное из них (например, ребро Start → A), совершаем соответствующий переход (выполняем соответствующую функцию перехода), тогда как оставшееся ребро Start→B заносится в стек. После перехода в A. Затем

из-за того, что всего доступна только одна вершина, переход происходит в вершину C.

Далее происходит ситуация, что из вершины опять выходит два ребра. Аналогичным образом для перехода выбираем, например, ребро C→E, тогда в стек заносится ребро C→F. После выполнения перехода в вершину \E ещё раз необходимо выполнение таких же действий. Допустим переход осуществляется по ребру E→D, а E→G заносится в стек. На текущий момент в стеке находится три ребра - Start→B, C→F, E→G.

После выполнения перехода в вершину D оказывается, что она требует для продолжения выполнение перехода ещё по ребру B→D. Следовательно, из стека достаётся ребро - E→G и осуществляется переход по данному ребру. Далее, при попытке переход через вершину G получается, что необходимо выполнения ребра D→G, достаётся следующее ребро из стека. Далее выполняется переход по ребру C→F. Аналогично переход по ребру F→H. Далее вершина H для продолжения обхода требует переход по G→H. Из-за этого берется ребро из стека и выполняется переход по нему.

Затем, после перехода по ребру B→D, в вершине D проверяется разрешается ли переход далее. Так как все нужные для продолжения обхода ребра уже пройдены, то переход разрешается. Далее из этой вершины по ребру D→G осуществляется переход. В вершине G повторяется аналогичная ситуация, поэтому осуществляется переход по ребру G→H. Далее по такому же правилу алгоритм доходит до вершины End, чем заканчивает обход [10].

## 2. Постановка задачи

Дано: ориентированный граф G (орграф), возможно, содержащий циклы, определённый множествами узлов $\{S_i\}_1^m$ и рёбер $\{e_{ij}: i \in [1..M], j \in [1..N]$, где M, N - некоторые целые числа.

Определены требование к графическому редактору.

1. Должна быть реализована возможность создавать ориентированный граф с “нуля”. А именно необходимо разработать возможности для добавления, удаления, редактирования рёбер и вершин графа.

2. Должна быть реализована возможность обхода построенного или загруженного графа.
3. Должна быть реализована возможность экспортировать созданный граф в файл формат aDOT.
4. Должна быть реализована возможность загрузить граф из файла формата aDOT.
5. Должна быть реализована возможность экспорта программного кода на языке программирования Python.
6. Графический редактор должен быть реализован в виде web-приложения.

В конечном итоге должен осуществляться визуализации и обход с выполнением python-кода по всему графу, который задан в файле формата aDOT или сделан через инструменты, предоставленные в web-приложении.

## 3. Программная реализация

Для разработки было принято решение использовать язык программирования Python. Для упрощения разработки функционала, связанного с графами, была выбрана библиотека NetworkX. Она предоставляет инструменты для создания, манипулирования и анализа графов и сетей, включая графы со сложной топологией, взвешенные и ориентированные графы, мультиграфы и гиперграфы.

Для реализации связи между клиентом и сервером был выбран Django - фреймворк для веб-разработки, написанный на языке Python. Он предоставляет программистам инструменты для создания веб-приложений быстро и эффективно. Django включает в себя множество функций, таких как ORM, автоматическую административную панель, систему маршрутизации URL, встроенный шаблонный движок и многое другое. Django также обеспечивает безопасность приложений с помощью встроенных инструментов защиты от CSRF-атак и инъекций SQL.

### 3.1. Требования к web-приложению

В финальной версии программы обязательно должны быть реализованы пять главных возможностей взаимодействий с web-сайтом:

1. Загрузить через web-сайт в базу данных граф, с которым будет производится взаимодействие. Загрузка производится путём нажатия специально сделанный для этого кнопкой с названием "Загрузить граф", после чего будет отображён граф, который был описан в загружаемом файле через алгоритм визуализации выбранный в результате исследования.
2. Для сохранения графа для дальнейшего взаимодействия с ним, должна быть реализована кнопка "Скачать". После нажатия данной кнопки должно начаться скачивание файла с названием "out.aDot", в котором на языке aDot будет представлен граф, с которым до момента нажатия данной кнопки работал пользователь.
3. Для пользователя необходимо предоставить возможность создание графа через инструменты, предоставленные на web-сайте. Среди инструментов обязательно должны быть представлены "Добавление узла", "Добавление ребра", "Удалить узел", "Удалить ребро", "Редактировать узел", "Редактировать ребро".
4. Должна быть представлена возможность обхода графа, предварительно полученного через загрузку из aDot файла или созданием из предоставленных специальных инструментов. Для этого должна быть кнопка "Начать обход".
5. На основной странице должен быть отображаться нарисованный граф или пустое место. Так же кнопки для реализации всех описанных до этого возможной взаимодействий с web-сайтом.

Необходимо отметить, что авторизация должна быть выполнена до захода на главную страницу. Другие пользователи системы не должны иметь доступ к данным друг друга, только к тем, что относятся непосредственно к ним.

### 3.2.Сценарии использование web-приложения

При разработке программы необходимо продумать все сценарии взаимодействия пользователя с web-приложением. Каждому модулю, кроме создания графа, относится по одному сценарию.

1. Сценарий, в котором реализуется модуль отображения графа:
2. Пользователей авторизуется заранее.
3. При заходе на главную страницу отображается разработанный им граф и все кнопки для взаимодействия с приложением.
4. Если граф не разработан, то отображается белый квадрат и все кнопки для взаимодействия с приложением.

Сценарий, в котором реализуется модуль скачивание файла, в котором будет граф в формате aDot:

1. Пользователей авторизуется заранее.
2. На главной странице нажимает кнопку "Скачать", после чего начинается загрузка файла с графом в формате aDot, с которым работал пользователь.
3. Если пользователь нажмёт кнопку "Скачать", когда граф ещё не создан, то будет скачан пустой файл.

Сценарий, в котором реализуется модуль загрузка графа через файл, в котором он представлен в формате aDot:

1. Пользователей авторизуется заранее.
2. При нажатии кнопки "Загрузить граф", пользователь будет перенаправлен на страницу с формой для загрузки файла.
3. После загрузки файла, пользователь нажимает кнопку "Загрузить" и он попадает на главную страницу с загруженным графом из файла.
4. Если пользователь нажал кнопку "Загрузить", но не загрузил файл, то тоже будет перенаправлен на главную страницу, где будет белый квадрат на месте изображения графа, что означает отсутствие графа.

Сценарий, в котором реализуется модуль обхода графа:

1. Пользователей авторизуется заранее.

2. При нажатии кнопки "Начать обход", пользователь будет перенаправлен на страницу с формой для загрузки файла с СВМ и выбором начальной и конечной вершиной. По умолчанию стоит из первой загруженной вершину в неё же, выбрать можно только из списка существующих вершин.
3. При загрузке файла с СВМ и выбором вершин будет выполнен обход графа и пользователь автоматически переходит на главную страницу.
4. Если файл загружен не будет, то выполнится обход без учёта СВМ.

Далее представлены сценарии, в котором реализуется модуль редактирования графа для каждой отдельной кнопки редактировании элементов графа.

Сценарий для добавления вершины:

1. Пользователей авторизуется заранее.
2. Пользователь на главной странице нажимает на кнопку "Добавить вершину". После чего попадёт на страницу с формой для добавления вершины, где надо будет выбрать название вершины и файл с функцией-селектором к ней.
3. После нажатия кнопки "Добавить", если все данные заполнены корректно, то пользователь будет переадресован на главную страницу с изменённым графом.
4. В случае, если указана повторная вершина, то пользователь будет переадресован на главную страницу и увидит не изменённый граф.
5. Если не загружен файл, то пользователь будет переадресован на главную страницу, а в вершине будет отсутствовать файл с функцией-селектором.

Сценарий для добавления ребра:

1. Пользователей авторизуется заранее.
2. Пользователь на главной странице нажимает на кнопку "Добавить ребро". После чего попадёт на страницу с формой для добавления ребра, где надо будет выбрать название начальной вершины,

конченой вершины, файл с функцией-обработчиком, файл с функцией-предикатом и reverse. У вершины будет доступен выбор только среди тех, кто уже существуют.

3. После нажатия кнопки "Добавить", если все данные заполнены корректно, то пользователь будет переадресован на главную страницу с изменённым графом.
4. Если не загружен файл с функцией-обработчиком, то пользователь будет переадресован на главную страницу, а в ребре будет отсутствовать файл с функцией-обработчиком.
5. Если не загружен файл функцией-предикатом, то пользователь будет переадресован на главную страницу, а в ребре будет отсутствовать файл с функцией-предикатом.

Сценарий для изменения вершины:

1. Пользователей авторизуется заранее.
2. Пользователь на главной странице нажимает на кнопку "Изменить вершину". После чего попадёт на страницу с формой для изменения данных вершины, где будет доступен список всех вершин из существующих и возможность загрузить файл с функции-селектором.
3. После нажатия кнопки "Изменить", если все данные заполнены корректно, то пользователь будет переадресован на главную страницу с изменёнными данными вершины.
4. Если файл с функцией-селектором не будет загружен, то пользователь будет переадресован на главную страницу с изменёнными данными вершины, в которой будет отсутствовать файл.

Сценарий для изменения ребра:

1. Пользователей авторизуется заранее.
2. Пользователь на главной странице нажимает на кнопку "Изменить ребро". После чего попадёт на страницу с формой для изменения данных ребра, где надо будет выбрать название ребра из

существующих, файл с функцией-обработчиком, файл с функцией-предикатом и reverse.

3. После нажатия кнопки "Изменить", если все данные заполнены корректно, то пользователь будет переадресован на главную страницу с изменёнными данными ребра.
4. Если файл с функцией-обработчиком не будет загружен, то пользователь будет переадресован на главную страницу с изменёнными данными ребра, в которой будет отсутствовать файл для функции-обработчиком, но для функции-предиката изменится на выбранный.
5. Если файл с функцией-предиката не будет загружен, то пользователь будет переадресован на главную страницу с изменёнными данными ребра, в которой будет отсутствовать файл для функции-предиката, но для функции-обработчиком изменится на выбранный.
6. Если файлы с функцией-предиката и с функцией-обработчиком не будет загружен, то пользователь будет переадресован на главную страницу с изменёнными данными ребра, в которой будет отсутствовать файл для функции-предиката и функции-обработчиком.

Сценарий для удаления вершины:

1. Пользователей авторизуется заранее.
2. Пользователь на главной странице нажимает на кнопку "Удалить вершины". После чего попадёт на страницу с формой для удаления вершины, где необходимо будет выбрать вершину из существующих.
3. После нажатии кнопки "Удалить", пользователь будет переадресован на главную страницу с изменённым графом.

Сценарий для удаления ребра:

1. Пользователей авторизуется заранее.

2. Пользователь на главной странице нажимает на кнопку "Удалить ребро". После чего попадёт на страницу с формой для удаления ребра, где необходимо будет выбрать ребро из существующих.
3. После нажатии кнопки "Удалить", пользователь будет переадресован на главную страницу с изменённым графом.

### 3.3. Архитектура, разрабатываемого ПО

В программе должно быть реализована три класса:

1. Graph: класс графа, с которым взаимодействует пользователь и который создаёт изображения графа, который он "содержит". Основной объект без взаимодействия, с которым невозможна работа с приложением
2. Node: класс вершины, который хранит всю информацию о вершине и из объектов этого класса, в классе графа создаются вершины графа, который будет отображён для пользователя в итоге.
3. Edge: класс вершины, который хранит всю информацию о ребре и из объектов этого класса, в классе графа создаются рёбра графа, который будет отображён для пользователя в итоге.

В классе Graph имеются следующий поля:

1. Edges: массив рёбер, которые хранятся в создаваемом графе. Поле необходимо для хранения объектов класса Edge, которые в свою очередь используются для сохранения данных о ребре, помимо начальной и конечной вершины.
2. Nodes: массив вершин, которые хранятся в создаваемом графе. Поле необходимо для хранения объектов класса Node, которые в свою очередь используются для сохранения данных о вершине.
3. Stack: техническое поле, являющиеся стеком, который используется при обходе графа.
4. Digraph: техническое поле, являющиеся объектом класса DiGraph из networkx. Необходимо для построения и создания изображения графа.

5. Core: файл с кодом СВМ, в котором хранятся данные, которые будут изменятся при обходе по графу.

В классе Node должны иметься следующий поля:

1. Name: название узла, для которого был создан объект данного класса.
2. h: файл, в котором представлен код функции-селектора, и который будет исполняться при обходе графа.
3. edge_continue: словарь вершин, которые необходимы для прохода в данную вершину. Является одним из двух инструментов, предназначенный для реализации синхронизации при обходе.

В классе Edge должны иметься следующий поля:

1. start: начальная вершина ребра, для которого был создан объект данного класса.
2. end: конечная вершина ребра, для которого был создан объект данного класса.
3. f: файл, в котором представлен код функции-обработчика, и который будет исполняться при обходе графа.
4. g: файл, в котором представлен код функции-предиката, и который будет исполняться при обходе графа.
5. reverse: булевая переменная, которая определяет необходимо ли это ребро для прохождения в конечную вершину.

В методах класса графа должны быть реализованы следующие методы для выполнения основных функция взаимодействия с web-приложением:

1. make_graph(name) - метод, который создаёт изображение и сохраняет в виде изображении в name.
2. read_from_aDot(name) - метод, читающей из файла с названием name (тип string) данные, затем сохраняет класс графа.
3. add_eage(edge) - метод, добавляющий ребро в класс графа.
4. remove_eage(edge) - метод, удаляющий ребро в классе графа
5. add_node(node) - метод, добавляет узел node
6. remove_node(node) - метод, удаляющий узел node

7. save_into_aDot(path) - метод, сохраняет граф в файл с названием out.aDot по пути path
8. run(start_node, end_node) - метод, позволяющий реализовать обход графа

Так же в данном классе должны быть представлены функции, которые необходимы для взаимодействия основных модулей web-приложения между друг другом:

1. get_node(name) - метод, позволяющий получить объект класса узла, по его названию, представленном в аргументе name.
2. get_edges(node_start, node_end) - метод, позволяющий получить объект класса ребра, по его начальной и конечной вершине, представленном в аргументах node_start и node_end, соотвественно.

В методах класса вершины должны быть реализованы:

1. execute(graph) - метод, который позволяет выполнить код селектора находящийся в этой вершине.

В методах класса ребра должны быть реализованы:

1. execute(graph) - метод, который позволяет выполнить код функции-обработчика и функции-предиката находящихся в этом ребре.

Краткое описание модулей и их взаимодействия через разработанные классы:

1. Открыть из файла - модуль для открытия и визуализации графа из файла формата aDot. Исполняется путём вызова метода read_from_aDot из объекта графа. Так же в вызванном методе впервые создаются все объекты классов ребра и вершины.
2. Создать новый граф - модуль ответственный за редактирование частей графа. Исполняется через методы add_eage, remove_eage, add_node, remove_node, вызываемые у объекта класса графа. Здесь при вызове каждого метода создания конструируется новый объект классов элемента графа, а для удаления достаточно названий.

3. Сохранение в файл - модуль ответственный за сохранения графа в файл типа aDot. Для использования необходимо вызвать метод save_into_aDot у объекта класса графа. При исполнении используются все классы для получения информации о графе из них. Блок-схема модуля представлена на рис. 6.


Рисунок 6. Строение реализации модуля ответственного за сохранения графа в Adot файл

4. Обход графа - модуль ответственный за обход графа. Для использования необходимо вызвать метод run у объекта класса графа. При исполнении в основном используется только он. Блок-схема модуля представлена на рис. 7.

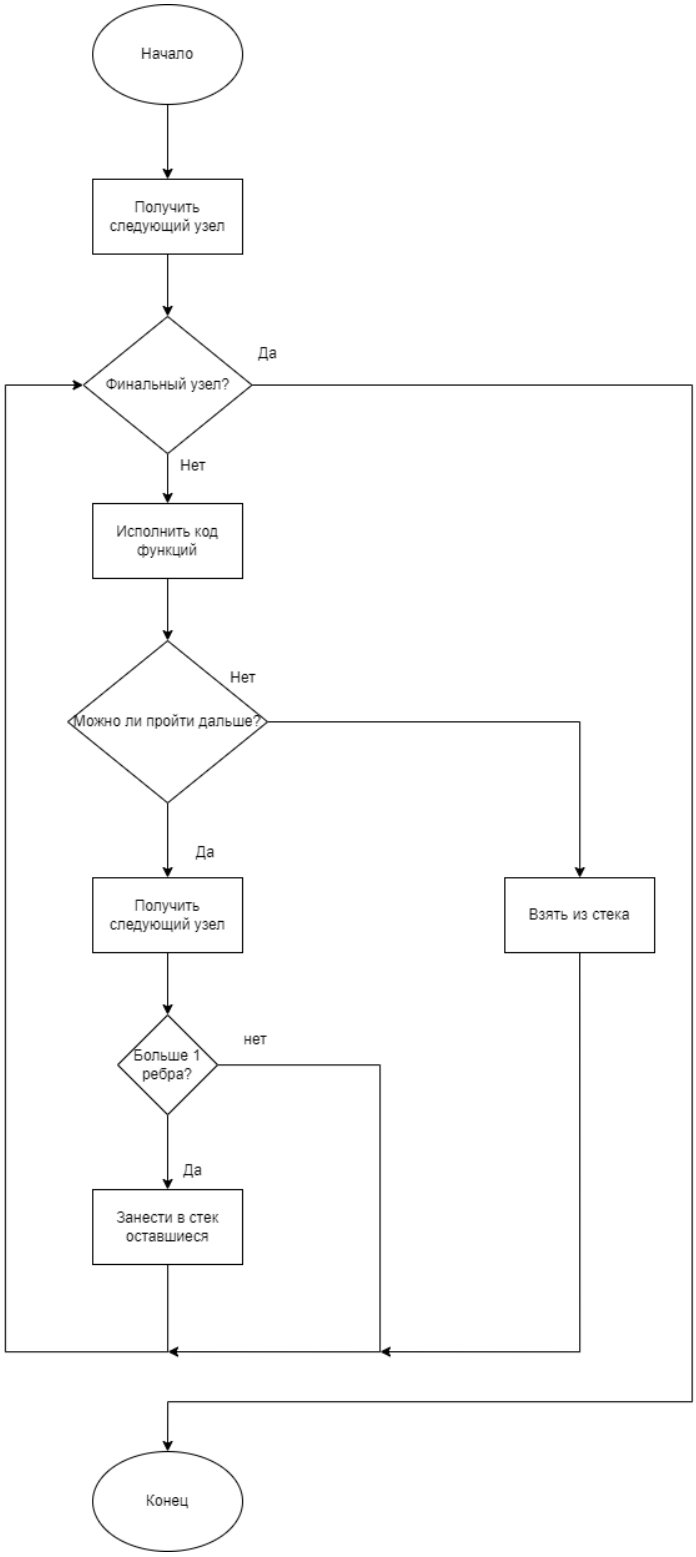


Рисунок 7. Строение реализации модуля ответственного за обход графа

5. Отобразить графа - модуль ответственный за создания изображения графа. Для использования необходимо вызвать метод make_graph у объекта класса графа. При исполнении используется только он.

### 3.4. Серверная часть web-приложения

Схема взаимодействия модулей, которые реализуют заданные пользовательские сценарии, представлена на рисунке 8.

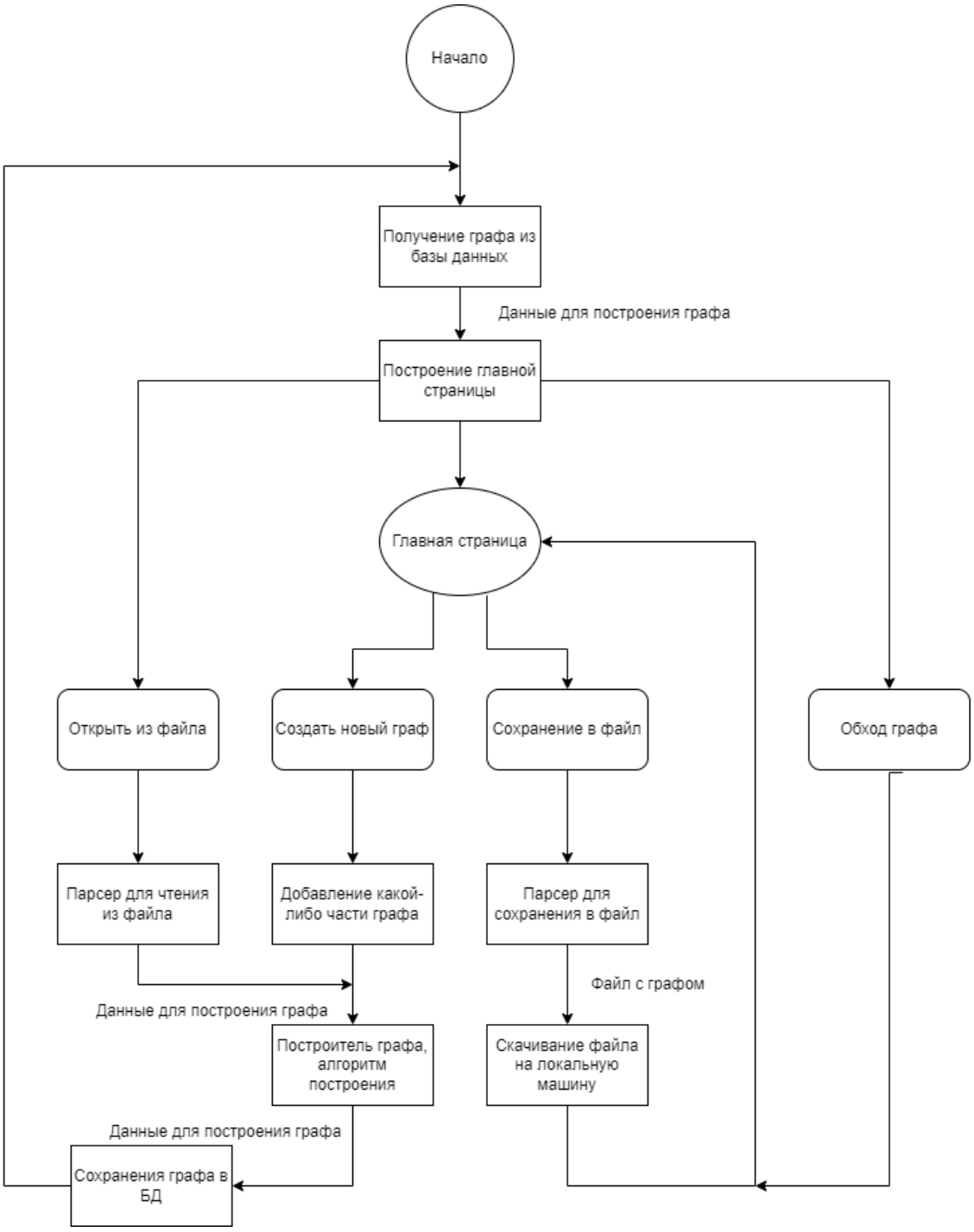


Рисунок 8. Схема модулей программы

Любой адрес возможно изменить в специальном файле с url-адресами, поэтому были выбраны временные названия. К каждому такому адресу соответствует своя функция-обработчик. Так же при заходе на адрес по умолчанию отправляется запрос, которые в данной программе бывают 2 типов: GET и POST. Обычно первым отправляется GET-запрос.

GET запрос - это один из методов передачи данных веб-серверу. Он используется для получения данных от сервера, которые затем отображаются на веб-странице. GET запрос передает данные в URL-адресе, который может быть виден в адресной строке браузера. В GET запросе данные передаются в виде параметров, разделенных знаком вопроса, и состоящих из имени параметра и его значения, разделенных знаком равно. GET запросы обычно используются для получения информации, такой как результаты поиска или содержимое страницы.

POST запрос - это метод передачи данных на веб-сервер. Он используется для отправки данных на сервер, которые затем обрабатываются и сохраняются. В отличие от GET запроса, данные в POST запросе передаются в теле запроса, а не в URL-адресе (как в GET-запросах). POST запросы часто используются для отправки форм, таких как регистрационные формы или формы заказа товаров. Данные в POST запросе могут быть зашифрованы, что делает их более безопасными для передачи конфиденциальной информации.

Для реализации сценария просмотра графа и главной страницы соответственно используется url-адрес "/". При переходе по этому адресу из базы данных создаётся изображение графа, принадлежащего зашедшему пользователю.

Для реализации сценария скачивания графа в формате aDot файла так же используется url-адрес "/". Нужный файл получается из базы данных и затем при нажатии кнопки "Скачать" автоматически производится скачивание.

Для реализации сценария считывания графа из aDot файла используется url-адрес "/read_adot". В зависимости от типа запроса меняется действия, исполняющиеся на этом адресе.

При использовании GET-запроса в ответ пользователю будет выдана форма для заполнения пользователя, в которой представлены все необходимые для продолжения работы поля. В данном случае просьба загрузить файл в формате aDot, в котором представлен новый граф.

При использовании POST-запроса в ответ пользователь будет переадресован (автоматически перейдёт) на url-адрес "/", а в программе очистится вся информация о старом графе и вместо неё будут данные нового графа, полученный из отправленного пользователем файла, который тоже сохранится в базу данных.

Для реализации сценария обхода графа используется url-адрес "/run". В зависимости от типа запроса меняется действия, исполняющиеся на этом адресе.

При использовании GET-запроса в ответ пользователю будет выдана форма для заполнения пользователя, в которой представлены все необходимые для продолжения работы поля. В данном случае просьба загрузить файл, в котором находится СВМ, и указание начальной и конечной вершины обхода графа.

При использовании POST-запроса в ответ пользователь будет переадресован на url-адрес "/", а в программе выполнится обход данных в соответствии с заданными параметрами.

Для реализации сценария создания графа используется url-адреса "add_node", "add_node", "add_edge", "add_edge", "edit_node", "edit_node", "edit_edge", "edit_edge", "remove_node", "remove_node", "remove_edge". В зависимости от типа запроса меняется действия, исполняющиеся на этих адресах.

При использовании GET-запроса в ответ пользователю будет выдана форма для заполнения пользователя, в которой представлены все необходимые для продолжения работы поля. Для каждого url-адреса разные данные, однако все они связаны с изменениями в графе.

При использовании POST-запроса в ответ пользователь будет переадресован на url-адрес "/", а в программе выполнится изменение в графе. После изменения в базу данных пересохранится изменённый элемент, так же

заново сохранится aDot файл, в котором будет представлен уже изменённый граф. Старый должен удалится.

### 3.5. Проектирование базы данных

Так как исходя из определённых ранее требований следует, что необходимо хранить файлы, то было принято решение хранить с использованием встроенной в Django базы данных SQLite3.

При проектировании системы базе данных было решено иметь 3 таблицы (рис.9):

1. Adot: в данной таблице хранится всё данные пользователя, которые он преобразует при работе с web-сайтом.
2. Node: таблица узлов, в которой хранится узлы и вся информация о них в графе, с которым в данный момент работает (или в предыдущий раз работал) пользователь. В ней же хранятся все файлы, в которых хранятся функции-селекторы.
3. Edge: таблица рёбер, в которой хранится рёбра и вся информация о них в графе, с которым в данный момент работает (или в предыдущий раз работал) пользователь. В ней же хранятся все файлы, в которых хранятся функции-предикаты и функции-обработчики.

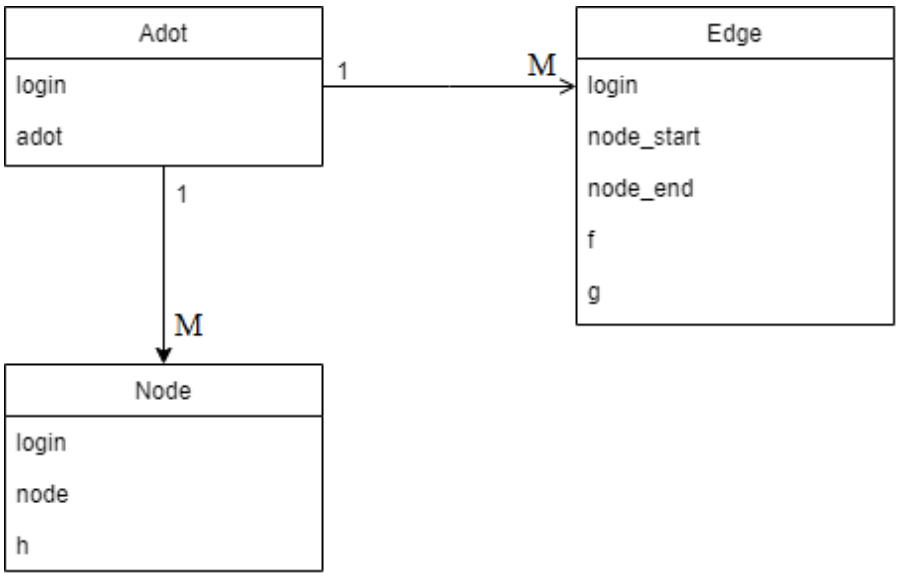


Рисунок 9. Архитектура базы данных

В таблице Adot находятся следующие поля:

1. login: логин пользователя для однозначной идентификации записей, чтобы выбирать данные доступные только для того, кто сейчас использует web-сайт.
2. adot: файл, в котором в формате aDot представлен граф, который будет отображаться для пользователя на главной странице или скачен через специальную кнопку на основной странице web-приложения.

В таблице Node находится следующие поля:

1. login: логин пользователя для однозначной идентификации записей, чтобы выбирать данные доступные только для того, кто сейчас использует web-сайт.
2. node: название вершины в графе, с который работает пользователь на главной странице
3. h: файл, в котором находится функция-селектор, определённая для вершины, к которой она принадлежит и записана в базе данных в этой же строке.

В таблице Edge находится следующие поля:

1. login: логин пользователя для однозначной идентификации записей, чтобы выбирать данные доступные только для того, кто сейчас использует web-сайт.
2. node_start: название начальной вершины ребра, которое сохранено в данной строке.
3. node_end: конечная вершина вершины ребра, которое сохранено в данной строке.
4. f: файл, в котором находится функция-обработчика, определённая для ребра, к которому она принадлежит и записана в базе данных в этой же строке.
5. g: файл, в котором находится функция-предикат, определённая для ребра, к которому она принадлежит и записана в базе данных в этой же строке.

# 4. Тестирование и отладка

## 4.1. Построение графа

Для тестирования построение графа через доступные кнопки в интерфейсе можно построить граф, представленный на рис. 10.


Рисунок 10. Построенный через приложение граф

В случае успешного прохождение теста на экране должен появится данный граф.

Далее, для тестирования удаления ребра из графа удаляется ребро из вершины 2 в вершину 3. В результате должен получится граф представленный на рисунке 11.


Рисунок 11. Граф с удалённым ребром

После для тестирования удаление вершины из графа удаляется вершина с названием “3”. В результате должен получится граф представленный на рисунке 12. Ребро, приходящее в эту вершину, тоже должно быть удалено.


Рисунок 12. Граф с удалённой вершиной

### 4.2. Взаимодействие файлами формата aDot

Для тестирования основного функционал и экспорта в файл с помощью доступных инструментов ранее сделанный граф необходимо скачать на компьютер при нажатии кнопки.

В результате скачивания должен получится файл с описание графовой модели в формате aDOT представленным на рис.13.

out.aDot – Блокнот

Файл Правка Формат Вид Справка

```
digraph TEST
{
// Parallelism
// Functions
f1 [module=None, entry_func=None]
// Predicates
p1 [module=None, entry_func=None]
// Edges
edge_1 [predicate=p1, function=f1]
// Graph model description
1 ->2 [morphism=edge_1, reverse=True]
2 ->3 [morphism=edge_1, reverse=True]
}
```

Рисунок 13. Полученное описание графовой модели в формате aDOT

Тестирование импорта графа производится путём загрузки файла из предыдущего пункта. В случае успешного выполнения получится граф представленный на рис. 10.

В случае успешной загрузки графа получим модель графа идентичную сделанной до этого.

### 4.3. Обход графа

Для проверки работоспособности необходимо сделать граф представленный на рис. 14 и загрузить специально созданный для этого файл, вершины и ребра, которое обеспечивает при проходе по каждой вершине прибавлять в специальную переменную единицу.

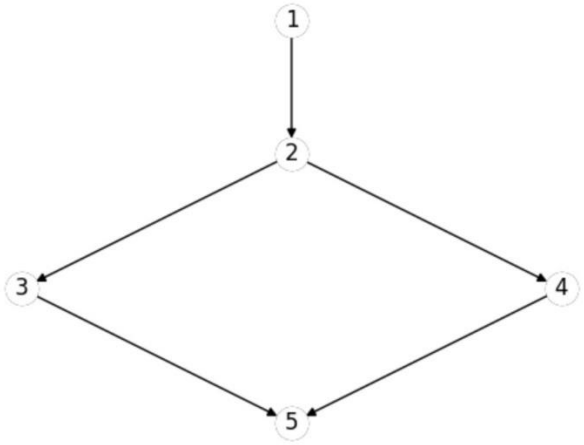


Рисунок 14. Граф для проверки обхода

При успешном проходе во всех вершинах будет цифра 1, что представлено на рисунке 15.


Рисунок 15. Результат тестирование обхода

# ЗАКЛЮЧЕНИЕ

Была проведена работа по изучению алгоритмов визуализации графов, выделены основные их виды. После проведения анализа алгоритма были выделены преимущества и недостатки каждого из них, после чего принято решение об использовании алгоритма dot.

Из-за наличия селекторов и необходимости обхода не по всему графу в ряде случаев было принято решение о создании собственного алгоритма.

По результатам применения разработанного ПО было выявлено, что оно удовлетворяет описанным ранее требованиям.

Программа разработана с учётом возможности дополнения каких-либо частей, а именно:

1. Добавления новых атрибутов вершин и дуг графа в файле формате aDot.
2. Интегрирование иных алгоритмов визуализации и обходов графов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. А. П. Соколов, А. Ю. Першин. “Система автоматизированного проектирования композиционных материалов. Часть 2. Вычислительная подсистема, распределенные вычисления с применением графоориентированного подхода” // Известия СПбГЭТУ «ЛЭТИ». – 2020- № 10 С. 49-52, 57
2. Пупырев С.Н. Тихонов А.В. “Визуализация динамических графов для анализа сложных сетей” // Моделирование и анализ информационных систем. 2010. Т. 17, № 1. С. 117 – 135.
3. Алгоритм Фрюхтермана-Рейнгольда. // pco.iis.nsk.su – URL: http://pco.iis.nsk.su/wega/index.php/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A4%D1%80%D1%8E%D1%85%D1%82%D0%B5%D1%80%D0%BC%D0%B0%D0%BD%D0%B0-%D0%A0%D0%B5%D0%B9%D0%BD%D0%B3%D0%BE%D0%BB%D1%8C%D0%B4%D0%B0 (дата обращения: 16.11.2022).
4. Алгоритм Камада-Кавай. // pco.iis.nsk.su – URL: http://pco.iis.nsk.su/wega/index.php/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D0%B0%D0%BC%D0%B0%D0%B4%D0%B0-%D0%9A%D0%B0%D0%B2%D0%B0%D0%B9 (дата обращения: 15.11.2022).
5. Построение графов: пошаговый гайд. // habr.com – URL: https://habr.com/ru/company/leader-id/blog/488058/ (дата обращения: 16.11.2022).
6. Метод укладки Gephi. Метод Фрухтермана-Рейнгольда и Openord. // newtechaudit.ru – URL: https://newtechaudit.ru/metod-ukladki-gephi-fruhtermana-rejngolda-i-openord/ (дата обращения: 16.11.2022).
7. Гамма-алгоритм. // neerc.ifmo.ru – URL: https://neerc.ifmo.ru/wiki/index.php?title=PYPөPёPёPөPөPњPγP«CЃPчCЂPё (дата обращения: 21.11.2022).

8. Kozo Sugiyama and Kazuo Misue. “A Simple and Unified Method for Drawing Graphs: Magnetic-Spring Algorithm” - Shizuoka, JAPAN - C. 364-375.
9. Emden R. Gansner Eleftherios Koutsofios Stephen C. North, Vo Gem-Phong. “A Technique for Drawing Directed Graphs” // EEE Transactions on Software Engineering 1993 Vol. 19, №.3, C 214-229.
10. Крехтунова Д., Ершов В., Муха В., Тришин И., Василян А. Р., Журавлев Н.В. Разработка систем инженерного анализа и ресурсоемкого ПО (rndhpc): Научно-исследовательские заметки. / Под редакцией Соколова А.П. [Электронный ресурс] - Москва: 2021. - 85 с. URL: https://arch.rk6.bmstu.ru (облачный сервис кафедры РК6)

## ПРИЛОЖЕНИЕ А

В графическую часть дипломного проекта входит:

1. Презентация PowerPoint.