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 «Робототехника и комплексная автоматизация»

КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Модели и методы анализа проектных решений»

на тему

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

Студент РК6-72Б (группа) _____ (подпись, дата) Журавлев Н.В. (ФИО)

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

Москва, 2022

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*

«____» _____ 2022 г.

ЗАДАНИЕ

на выполнение курсового проекта

Студент группы: РК6-72Б

Журавлев Николай Вадимович

(фамилия, имя, отчество)

Тема курсового проекта: Разработка web-приложения, обеспечивающего построение графовых моделей алгоритмов обработки данных

Источник тематики (кафедра, предприятие, НИР): кафедра

Тема курсового проекта утверждена на заседании кафедры «Системы автоматизированного проектирования (РК-6)», Протокол № _____ от «____» _____ 2022 г.

Техническое задание

Часть 1. *Аналитический обзор литературы.*

Более подробная формулировка задания. Следует сформировать, исходя из исходной постановки задачи, предоставленной руководителем изначально. Формулировка включает краткое перечисление подзадач, которые требовалось реализовать, включая, например: анализ существующих методов решения, выбор технологий разработки, обоснование актуальности тематики и пр. Например: «Должен быть выполнен аналитический обзор литературы, в рамках которого должны быть изучены вычислительные методы, применяемые

для решения задач кластеризации больших массивов данных. Должна быть обоснована актуальность исследований.»

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

Более подробная формулировка задания. Формулировка заголовка части может отличаться от работы к работе (например, может быть просто «Математическая постановка задачи» или «Архитектура программной реализации»), что определяется конкретной постановкой задачи. Содержание задания должно детальнее и обязательно конкретно раскрывать заголовок. Например: «Должна быть создана математическая модель распространения вирусной инфекции и представлена в форме системы дифференциальных уравнений».

Часть 3. *Проведение вычислительных экспериментов, тестирование.*

Более подробная формулировка задания. Должна быть представлена некоторая конкретизация: какие вычислительные эксперименты требовалось реализовать, какие тесты требовалось провести для проверки работоспособности разработанных программных решений. Формулировка задания должна включать некоторую конкретику, например: какими средствами требовалось пользоваться для проведения расчетов и/или вычислительных эксперименто. Например: «Вычислительные эксперименты должны быть проведены с использованием разработанного в рамках ВКР программного обеспечения».

Оформление курсового проекта:

Расчетно-пояснительная записка на 30 листах формата А4.

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

количество: 6 рис., 0 табл., 10 источн.
[здесь следует ввести по пунктам наименования чертежей, графических материалов, плакатов]

Дата выдачи задания «*10*» октября 2022 г.

Студент _____ Журавлев Н.В.
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

курсовой проект: 30 с., 6 рис., 0 табл., 10 источн.

ОРИЕНТИРОВАННЫЕ ГРАФЫ, ОБХОД ГРАФА, ВИЗУАЛИЗАЦИЯ ГРАФА, ADOT, WEB-ПРИЛОЖЕНИЕ.

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: курсовой проект.

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;

СОКРАЩЕНИЯ

This document is incomplete. The external file associated with the glossary ‘abbreviations’ (which should be called `output.gls-abr`) hasn’t been created.

Check the contents of the file `output.glo-abr`. If it’s empty, that means you haven’t indexed any of your entries in this glossary (using commands like `\gls` or `\glsadd`) so this list can’t be generated. If the file isn’t empty, the document build process hasn’t been completed.

You may need to rerun LaTeX. If you already have, it may be that TeX’s shell escape doesn’t allow you to run makeindex. Check the transcript file `output.log`. If the shell escape is disabled, try one of the following:

– Run the external (Lua) application:

```
makeglossaries-lite "output"
```

– Run the external (Perl) application:

```
makeglossaries "output"
```

Then rerun LaTeX on this document.

This message will be removed once the problem has been fixed.

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 (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 невязки при варьирование 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.

Для решения подзадачи визуализации графа необходимо определить, что известно о графе, который будет применяться. Дан ориентированный граф, возможно содержащий циклы и селекторы, так же переходы из узла самого в себя.

Для визуализации графов существует 2 способа визуализации графов:

1. Силовые алгоритмы визуализации графов - расположить узлы графа так, чтобы все рёбра имели более-менее одинаковую длину, и свести к минимуму число пересечений рёбер путём назначения сил для множества рёбер и узлов основываясь на их относительных положениях, а затем путём использования этих сил либо для моделирования движения рёбер и узлов, либо для минимизации их энергии.

2. Послойное рисование графа - это способ, в котором вершины ориентированного графа рисуются горизонтальными рядами или слоями с рёбрами, преимущественно направленными вниз.

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
– Multidimensional scaling - вершины являются силой, а рёбра уже являются пружинами
– Energy-Based - в этом подходе пытаются описать потенциальную энергию системы и найти положение вершин, которое будет соответствовать минимуму

Для решения поставленной задачи не имеет значения к какой категории принадлежит исследуемый алгоритм, поэтому необходимо рассмотреть самые часто упоминаемые, а именно: Fruchterman-Reingold, Алгоритм Идеса, Kamada Kawaii, Force Atlas 2, OpenOrd, Гамма – алгоритм.

Чтобы выбрать нужный для данного случая алгоритм необходимо выделить свойства результирующего графа, по котором будет произведён отбор:

– Граф является ориентированным
– Конец графа должен находится в противоположной стороне от начала
– Максимально допустимый граф является среднего размера
– В графе могут не иметься циклы

В алгоритме Fruchterman-Reingold используется пружинная физическая модель, где вершины определяются как тела системы, а ребра как пружины. Силы могут действовать только на вершины, вес пружин при этом не учитывается. Данные алгоритм не подходит, так как он не предусмотрен для ориентированных графов [2].

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

ходит, так как имеет самую высокое время работы из рассматриваемых - $O(V^3)$ [3].

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. Основная его задача - визуализация графов, в которых имеются подмножества с высокой степенью взаимодействия, таким свойством целевой граф не обладает [4].

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. Итерации контролируются через алгоритм имитации отжига. Предназначен для визуализации больших графов [5], следовательно не подходит для целевого графа, так как он является графом маленького или среднего размера.

Гамма - алгоритм основная идея которого заключается в том, что выделяются сегменты, затем в минимальном выделяется цепь, которая укладывается в любую грань, вмещающую данный сегмент [6]. Не подходит, так как в целевом графе может не оказаться циклов.

Алгоритм Идеса (Magnetic-Spring Algorithm) рёбра назначаются магнитной пружиной, и затем при воздействии магнитных полей вершины перемешаются [7]. Лучше всего для целевого графа.

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 был выбран алгоритм из этого типа. Исходя из этого для решения задачи визуализации графа, бы выбран алгоритм dot.

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[8].

Для решения задачи обхода графа из-за специфичности задачи было принято решение о создании собственного алгоритма. Особенностями больше всего влияющие на решение создания алгоритма являются - наличие селекторов, проход не всегда по всему графу.

Далее при описании алгоритма используется терминология, введённая в работах [9, 10].

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. А.1), сопоставленной с узлом S_i, результат выполнения которой определяет подмножество $\hat{E}_i = \{a_{ik} : a_{ik} \in E_i, {}_k(h_i(D_i)) = 1, D_i \multimap S_i\}$, где S_i – состояние данных ??, 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.

Замечание 1. Реализация $\hat{E}_i$ для каждого i возможна путём формирования структуры данных стек. Заполнение соответствующего стека элементами может быть осуществлено в произвольном порядке[1].

Для реализации требуемого алгоритма необходимо: сделать класс дуги и в класс узла добавить поля, которые определяют необходимое число для перехода через данный узел и количество дуг, которые уже пришли в узел. Класс дуги содержит два поля: начальный узел и конечный узел.

Рассмотрим работу алгоритма на примере орграфа, представленного на рисунке Б.1. Начальная вершина называется Start, а конечная End. Селекторы в данном примере отсутствуют.

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[2], 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.

[1]Должны быть выполнены все функции перехода F_{ij}, связанные с соответствующими рёбрами, в произвольном порядке (возможно в многопоточном режиме).

[2]Выполняем соответствующую функцию перехода при «боевом »режиме обхода.

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.

В разработанном А.П. Соколовым и А.Ю. Першиным графоориентированном программном каркасе, для описания графовых моделей был разработан формат aDOT, который является расширением формата описания графов DOT.

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 Постановка задачи

1.1 Концептуальная 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.

Разрабатываемый графический редактор должен:

1. Предоставлять возможность создавать ориентированный граф с нуля (добавление/удаление ребра/узла)
2. Предоставлять возможность обхода графа
3. Предоставлять возможность экспортировать созданный граф в формат aDOT
4. Предоставлять возможность загрузить граф из формата aDOT

Так же должен удовлетворять следующим требованиям:

– Иметь возможность экспорта программного кода на языке программирования Python

– Являться плагином для comwpc

В конечном итоге выполнения должен осуществляться визуализаци и обход с выполнением python-кода по всему графу, который, возможно, задан в файле формата aDOT.

2 Программная реализация

2.1 Архитектура

Для разработки было принято решение использовать язык программирования Python3.10 и библиотеку networkx для работу с графом.

В реализации программы должны быть реализованы четыре главных сценария действия: Открыть из файла, создание графа, сохранение в файлы, найти цикл. Так же необходимо создать простую базу данных для хранения графов для каждого пользователя. Общий вид схемы модулей представлен на рисунке В.1. В программе должно быть реализована 2 класса - Graph (класс графа) и Node (класс вершина).

В классе узла должны иметься следующий поля: словарь с ключом в виде следующих узлов и код при переходе к следующему узлу, являющимся значением являющейся строкой; название для узла.

Краткое описание модулей и их взаимодействия:

- Открыть из файла - модуль для открытия и визуализации графа из файла формата aDot.
- Создать новый граф - модуль ответственный за редактирование частей графа.
- Сохранение в файл - модуль ответственный за сохранения графа в файл типа aDot.
- Обход цикл - модуль ответственный за обход цикла в графе.

В классе графа в полях должно иметься массив класса узлов, в методах должны быть реализованы:

- make_graph(name) - метод, которые преобразует массив в объекте класса в переменную, которую можно сохранить в виде изображения с выбранным расширением.
- read_from_aDot(name) - метод, читающей из файла с названием name (тип string) данные, затем сохраняет их в поле для узлов.
- search_cicles() - метод, который находит циклы в графе, возвращает все циклы в виде строки формата "a-b-c[перенос строки]b-d-e[перенос строки]".

– add_eage(e1, e2) - метод, добавляющий ребро между двумя узлами из e1 (тип класса узла) в e2 (тип класса узла)

– delete_eage(e1, e2) - метод, удаляющий ребро между двумя узлами из e1 (тип класса узла) в e2 (тип класса узла)

– add_node(node) - метод, добавляет узел node

– delete_node(node) - метод, удаляющий узел node

– save_into_aDot(name) - метод, сохраняет граф в файл с названием name (тип string)

В базе данных необходимо иметь одну таблицу с двумя полями: пользователь и его граф.

3 Тестирование и отладка

3.1 Экспорт графа в aDOT файл

Для тестировании основная функционал и экспорта в файл с помощью доступных инструментов сделаем граф показанный на рисунке(Г.1).

В результате должен получиться файл с описание графовой модели в формате aDOT представленным на листинге (3.1).

Листинг 3.1. *Полученное описание графовой модели в формате aDOT*

```
digraph TEST
{
// Parallelism
s1 [parallelism=threading]
s2 [parallelism=threading]
// Function
f1 [module=f1_module, entry_func=f1_function]
// Predicates
p1 [module=p1_module, entry_func=p1_function]
// Transition
edge_1 [predicate=p1, function=f1]
// Graph model
__BEGIN__ -> s1
s1 => s2 [morphism=edge_1]
s1 => s3 [morphism=edge_1]
s2 => s4 [morphism=edge_1]
s2 => s5 [morphism=edge_1]
s3 -> s7 [morphism=edge_1]
s4 -> s6 [morphism=edge_1]
s5 -> s6 [morphism=edge_1]
s6 -> s9 [morphism=edge_1]
s7 -> s8 [morphism=edge_1]
s8 -> s9 [morphism=edge_1]
s9 -> __END__
}
```

3.2 Импорт из aDOT файла

Тестирование производится путём импорта файла из предыдущего пункта. В случае успешного выполнения получится граф представленный на рисунке (Д.1). Стоит отметить, что внешний вид изменится за счёт применения алгоритма визуализации.

3.3 Тестирование циклов в графе

В описание графовой модели на предыдущем шаге добавим несколько циклов и обратных ребер. Получим следующее описание графовой модели представленное на листинге (3.2).

Листинг 3.2. *Пример описание графовой модели в формате aDOT включающей циклы*

```
digraph TEST
{
// Parallelism
s7 [parallelism=threading]
s11 [parallelism=threading]
s2 [parallelism=threading]
s10 [parallelism=threading]
s1 [parallelism=threading]
// Function
f1 [module=f1_module, entry_func=f1_function]
// Predicates
p1 [module=p1_module, entry_func=p1_function]
// Transition
edge_1 [predicate=p1, function=f1]
// Graph model
__BEGIN__ -> s1
s4 -> s6 [morphism=edge_1]
s5 -> s6 [morphism=edge_1]
s7 => s6 [morphism=edge_1]
s7 => s1 [morphism=edge_1]
s11 => s8 [morphism=edge_1]
s11 => s2 [morphism=edge_1]
s12 -> s8 [morphism=edge_1]
s2 => s4 [morphism=edge_1]
s2 => s5 [morphism=edge_1]
```

```
26 s2 => s7 [morphism=edge_1]
27 s10 => s11 [morphism=edge_1]
28 s10 => s12 [morphism=edge_1]
29 s1 => s2 [morphism=edge_1]
30 s1 => s10 [morphism=edge_1]
31 s6 -> s9 [morphism=edge_1]
32 s8 -> s9 [morphism=edge_1]
33 s9 -> s6 [morphism=edge_1]
34 s9 -> __END__
35 }
```

В случае успешной загрузки графа получим модель графа, представленной на рисунке (E.1).

4 Анализ результатов

По результатам тестирования можно сделать вывод о работоспособности программы по части базовых функций. Предоставленные графы успешно обходятся. Формирование из aDot представляет собой удобочитаемый граф. Запись в aDot файл происходит в соответствии требованиям для этого формата.

Технически предусмотрена возможность добавления обработки новых атрибутов вершин и дуг графа в файле формате aDot. Потенциально можно добавить несколько алгоритмов визуализации или обхода графов.

На текущей стадии разработки программа не является web-приложение, что планируется исправится в дальнейшем. Так же несмотря на присутствие "заглушек"для исполнения Python-кода, сам функционал его исполнения и добавления, пока не создан.

ЗАКЛЮЧЕНИЕ

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

Разобраны виды обхода графов, однако из-за специфичности задачи было принято решение о создании собственного алгоритма. Особенностями больше всего влияющие на решение создания алгоритма являются - наличие селекторов, проход не всегда по всему графу.

По результатам тестирования выявлено, что программы выполняет большинство требуемых базовых функций.

Программа сделана с учётом на будущее дополнения каких-либо частей, а именно:

- Добавления новых атрибутов вершин и дуг графа в файле формате aDot.
- Использование множества алгоритмов визуализации графов с возможностью выбора определённого.
- Добавление нескольких алгоритмов обхода графов и возможность выбора какой из них будет использоваться для данного графа.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 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.

2 Алгоритм Фрюхтермана-Рейнгольда. // pco.iis.nsk.su – URL: http://pco.iis.nsk.su/wega/index.php/РѲРњРуР«СЃРчСЪРё_РhСЃСЊСЖСЪРҳСЃРёРѳР,,Рѳ-РгРҳРєР,,РуР«РњСÆРұРѳ.

3 Алгоритм Камада-Кавай. // pco.iis.nsk.su – URL: http://pco.iis.nsk.su/wega/index.php/РѲРњРуР«СЃРчСЪРё_РЭРѳРёРѳРұРѳ-РЭРѳРўРѳРє.

4 Построение графов: пошаговый гайд. // habr.com – URL: https://habr.com/ru/company/leader-id/blog/488058/.

5 Метод укладки Gephi. Метод Фрухтермана-Рейнгольда и Openord. // newtechaudit.ru – URL: https://newtechaudit.ru/metod-ukladki-gephi-fruhtermana-rejngolda-i-openord/.

6 Гамма-алгоритм. // neerc.ifmo.ru – URL: https://neerc.ifmo.ru/wiki/index.php?title=РУРѳРёРёРѳ-РѳРњРуР«СЃРчСЪРё.

7 A Simple and Unified Method for Drawing Graphs: Magnetic-Spring Algorithm. // link.springer.com – URL: https://link.springer.com/content/pdf/10.1007/3-540-58950-3_391.pdf.

8 Emden R. Gansner Eleftherios Koutsofios Stephen C. North, Vo Gem-Phong. A Technique for Drawing Directed Graphs. 1993.

9 A.P. Sokolov and A.Yu. Pershin. Graph-Based Software Framework for Implementation of Complex Computational Methods // Programming and Computer Software. 2019. Vol. 45, no. 5. P. 257—267.

10 Соколов А.П., Голубев В.О. Система автоматизированного проектирования композиционных материалов. Часть 3: графоориентированная методология разработки средств взаимодействия пользователь-система // Известия СПбГЭТУ «ЛЭТИ». 2021. № 2. С. 43–57.

Выходные данные

Журавлев Н.В.. Разработка web-приложения, обеспечивающего построение графовых моделей алгоритмов обработки данных по дисциплине «Модели и методы анализа проектных решений». [Электронный ресурс] — Москва: 2022. — 30 с. URL: `https://sa2systems.ru:88` *(система контроля версий кафедры РК6)*

Постановка:  *Доцент, к.ф.-м.н., Соколов А.П.*

Решение и вёрстка:  *студент группы РК6-72Б, Журавлев Н.В.*

2022, осенний семестр

A

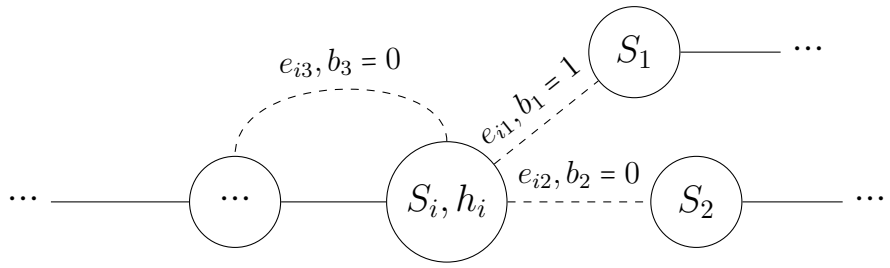

Рисунок А.1. Организация ветвления с помощью функции-селектора h_i из состояния S_i, где $\forall D_i \multimap\!\!\bullet\, S_i \rightrightarrows \{S_j\}_{j=1}^{n}$ и $h_i(D_i) = b = (b_1,b_2,b_3,\ldots,b_n) = (1,0,0,\ldots)$. Очевидно, что для представленной иллюстрации $\hat{E}_i = \{e_{i1}\}$ и переход по ветвям, для которых $b_i = 0$, запрещён

Б

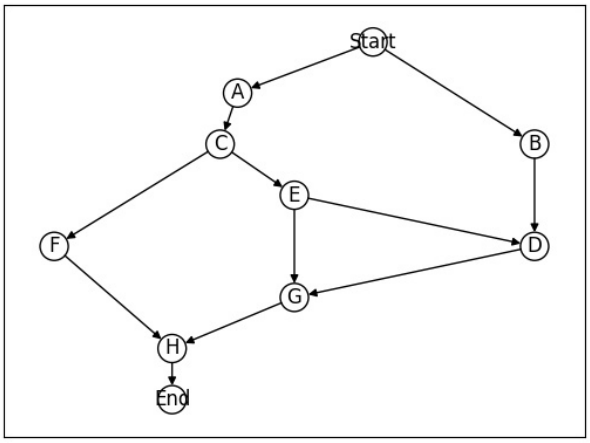

Рисунок Б.1. Пример орграфа без циклов

В

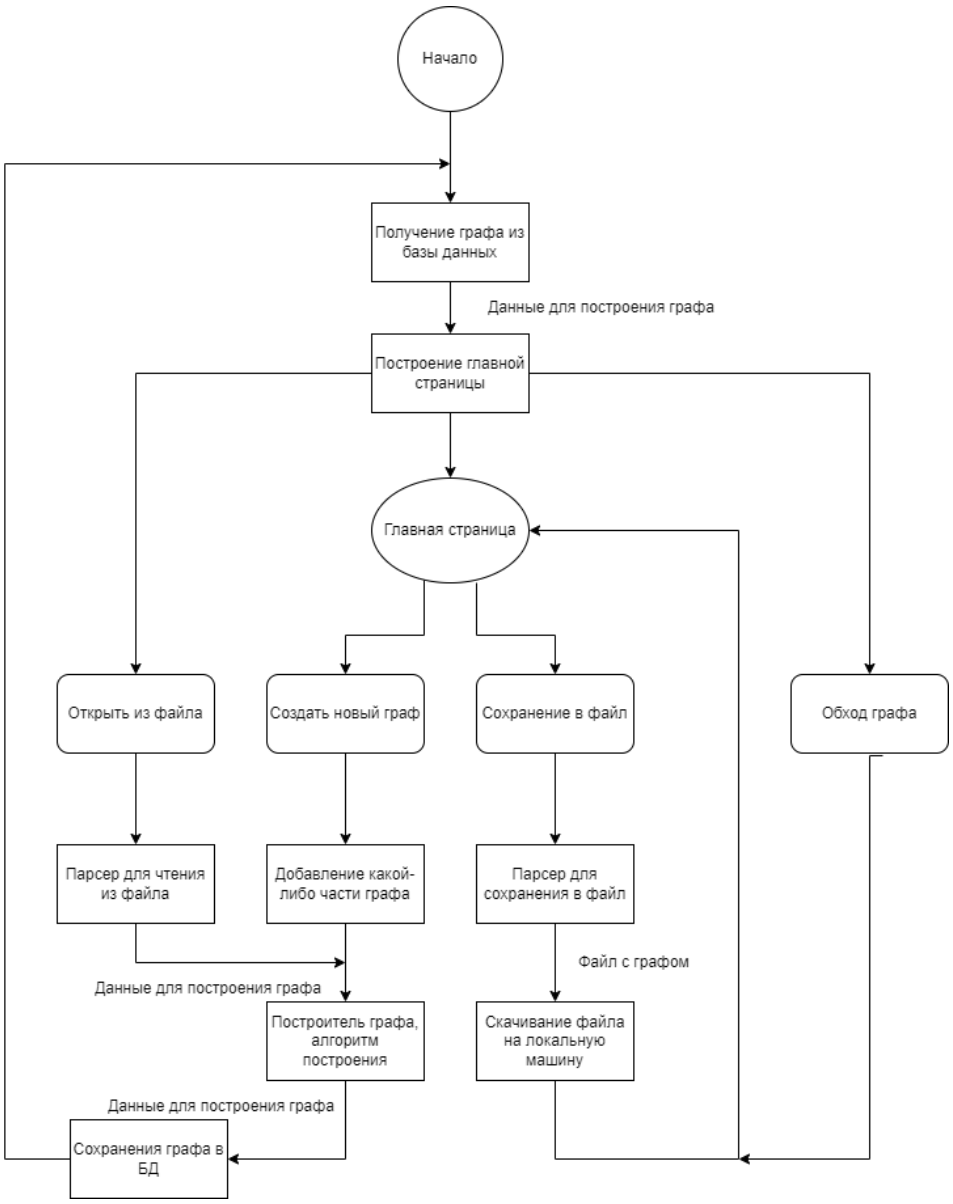

Рисунок В.1. Схема модулей программы

Г

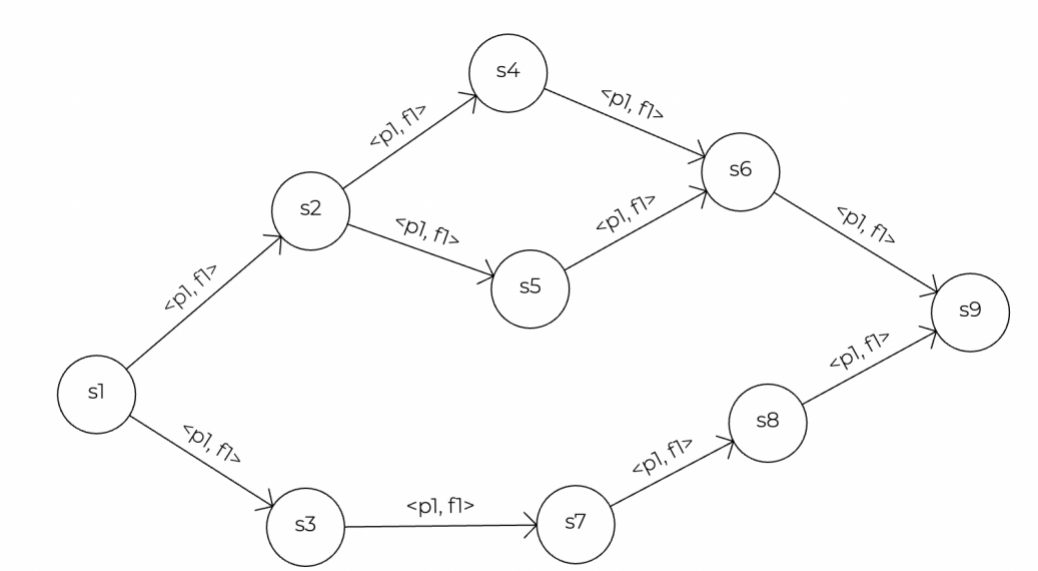

Рисунок Г.1. Ориентированный граф созданный с нуля в редакторе

Д

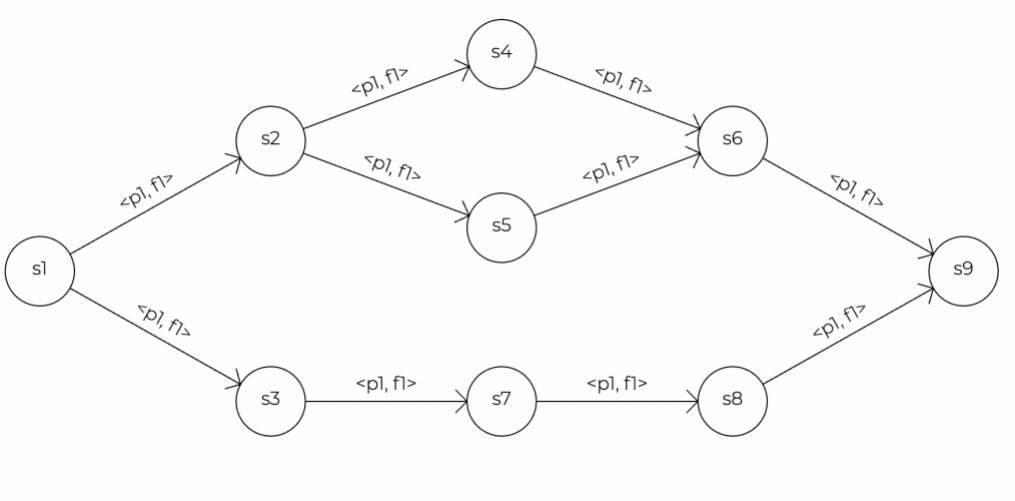

Рисунок Д.1. Загруженный в формате aDOT граф

Е

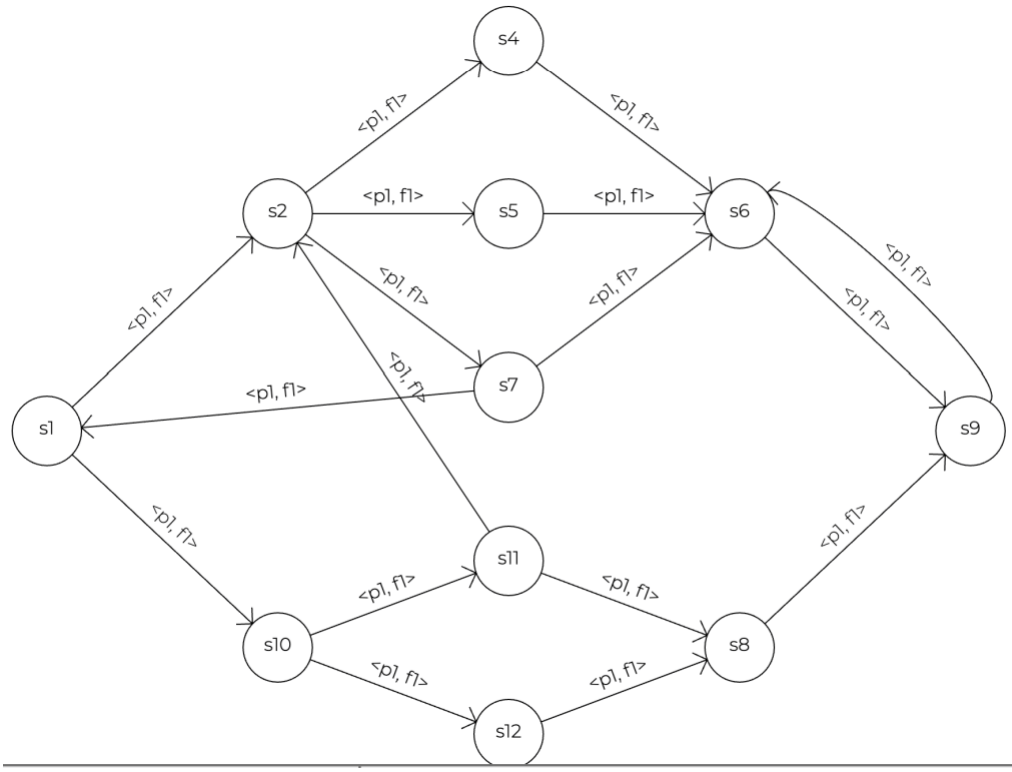

Рисунок Е.1. Загруженный в формате aDOT граф