МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Минимальное остовное дерево: алгоритм Борувки

Студент гр. 0382 _____ Корсунов А.А.

Студент гр. 0382 _____ Самулевич В.А.

Студент гр. 0383 _____ Сабанов П.А.

Руководитель _____ Фирсов М.А.

Санкт-Петербург

2022

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Корсунов А.А. группы 0382

Студент Самулевич В.А. группы 0382

Студент Сабанов П.А. группы 0383

Тема практики: Минимальное остовное дерево: алгоритм Борувки

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: алгоритм Борувки

Сроки прохождения практики: 29.06.2022 – 12.07.2022

Дата сдачи отчета: 12.07.2022

Дата защиты отчета: 12.07.2022

Студент _____ Корсунов А.А.

Студент _____ Самулевич В.А.

Студент _____ Сабанов П.А.

Руководитель _____ Фирсов М.А.

АННОТАЦИЯ

Целью учебной практики является освоение языка программирование «Java» путем разработки графического приложения для нахождение минимального остовного дерева графа с помощью алгоритма Борувки на этом языке. Разработка выполнялась в бригаде из трех человек, где каждый отвечал за определенную часть работы.

SUMMARY

The purpose of the training practice is to master the programming language "Java" by developing a graphical application for finding the minimum spanning tree of a graph using the Boruvka algorithm in this language. The development was carried out in a team of three people, where each was responsible for a certain part of the work.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель учебной практики — освоение языка программирования «Java». Итогом командной практической работы служит приложение, визуализирующее работу алгоритма Борувки — нахождение минимального остовного дерева графа. Пользователю приложения должны быть предоставлены возможность задать граф как посредством файла, так и с помощью графического интерфейса, а также возможность просмотра исполнения алгоритма по шагам с пояснениями происходящего в виде комментариев в самом приложении, либо же получение результата без промежуточных шагов.

Алгоритм Борувки (или алгоритм Борувки — Соллина) — это жадный алгоритм нахождения минимального остовного дерева в взвешенном связанном неориентированном графе. Впервые был опубликован в 1926 году Отакаром Борувкой в качестве метода нахождения оптимальной электрической сети в Моравии. Является первым зарегистрированным алгоритмом нахождения минимального остовного дерева.

Применение алгоритма:

Алгоритм Борувки решает задачу минимального остовного дерева. Примером такой задачи может являться поиск способа соединения городов дорогами так, чтобы их общая длина или стоимость была минимальной. Также задача встречается в компьютерных сетях при нахождении лучшего способа соединения узлов сети так, чтобы суммарная скорость соединения между ними была максимальной.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ (СПЕЦИФИКАЦИЯ)

1.1. Исходные Требования к программе

1.1.1. Постановка задачи.

Программа иллюстрирует алгоритм Борувки на заданном графе;

1.1.2. Формат входных данных.

Пользователю предлагается два варианта задания графа:

а) С помощью матрицы смежности, записанной в файл: в интерфейсе программы пользователю будет предложена опция загрузки этого файла, программа извлечет из него данные, проверит их корректность, и построит соответствующий матрице граф. Вершины графа после загрузки будут распределяться по кругу;

б) С помощью инструментов графического интерфейса:

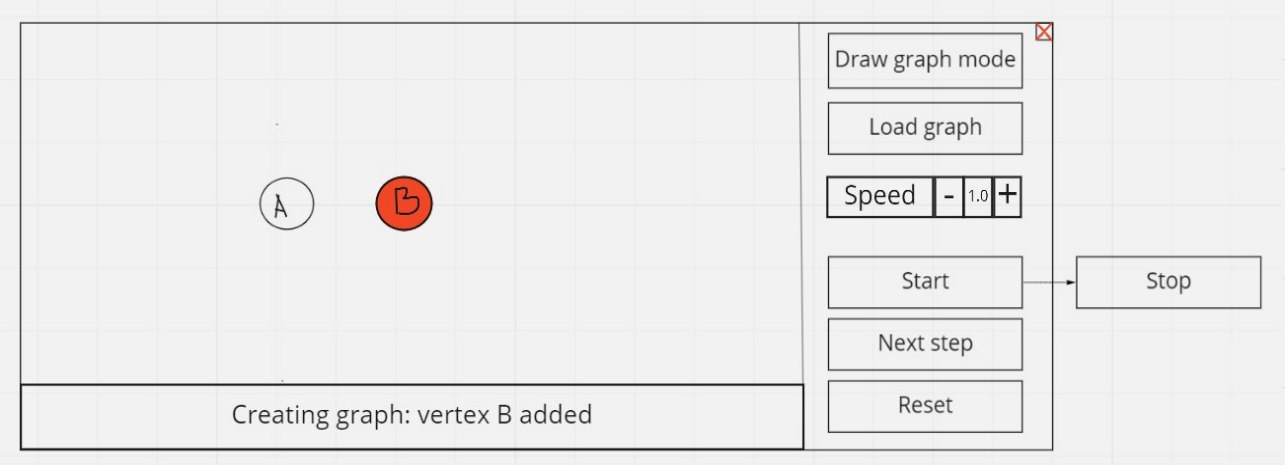


Рисунок 1 — Эскиз графического интерфейса

*Создание графа происходит в режиме «Draw graph mode» (он активируется после нажатия соответствующей кнопки в окне программы): добавление и удаление вершин происходит при нажатии левой и правой клавиши мыши соответственно.

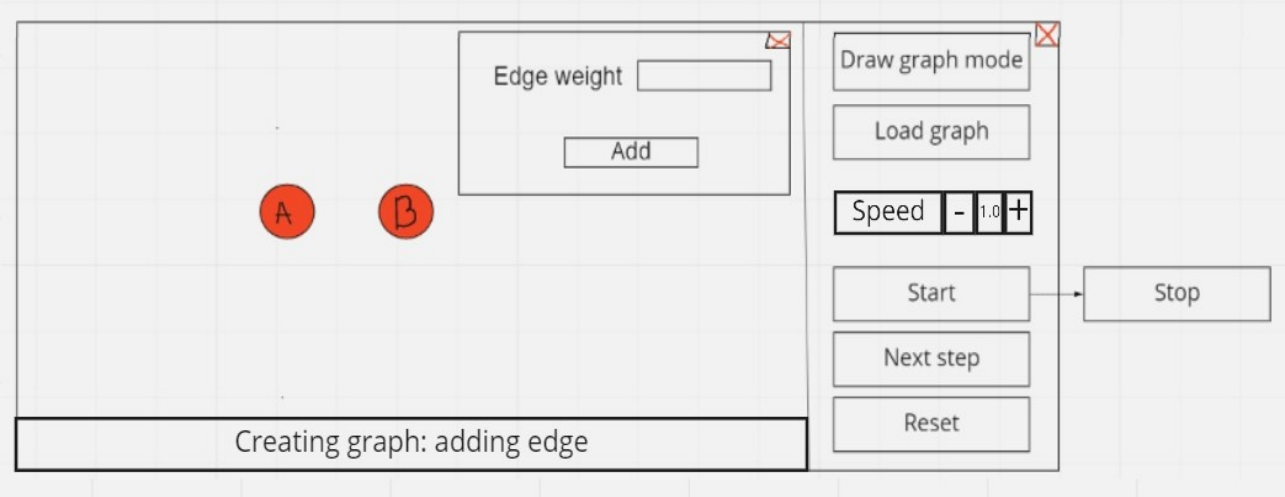


Рисунок 2 — Эскиз добавление и соединение вершин

*Для того чтобы соединить две вершины, нужно в «Draw graph mode» выбрать их по очереди с помощью левой клавиши мыши, после чего откроется диалоговое окно, в котором будет предложено указать вес нового ребра и подтвердить его добавление (как и вершины, ребра так же удаляются нажатием на них правой клавишей мыши).

1.1.3. Формат выходных данных.

После нажатия кнопки «Start» начинается пошаговая визуализация работы алгоритма. На каждом шаге различные компоненты связности выделяются каждая своим цветом. Выбранное на каждом «маленьком» шаге ребро окрашивается в красный цвет. Остановить визуализацию на конкретном шаге можно будет с помощью кнопки «Stop». Также, для перемещения между шагами, будет создана кнопка «Next step»;

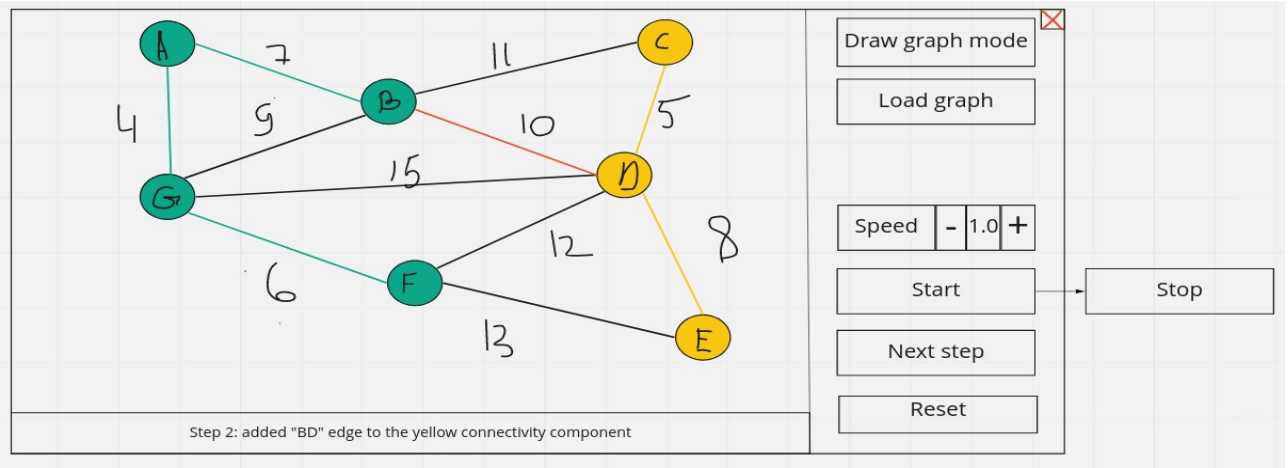


Рисунок 3 — Эскиз визуализации шага алгоритма Борувки

Кнопка «Next step» будет перемещать алгоритм на один «маленький шаг»: шаг, при котором соединяются две компоненты связности;

Также в приложении есть текстовое поле, в котором отображаются шаги алгоритма: «маленькие» и «большие» («маленький» шаг — шаг, при котором ребром соединяются две компоненты связности, «большой» шаг — шаг, при котором число компонент связности уменьшается как минимум вдвое);

К тому же, минимальное остовное дерево будет записано в виде матрицы в текстовом поле в конце работы алгоритма. Информацию из текстового поля можно будет копировать. По этому полю можно будет передвигаться сверху-вниз и снизу-вверх с помощью колеса мыши.

1.1.4. Дополнительные возможности графического интерфейса.

*Вне режима Draw graph mode можно будет передвигать вершины графа (зажав левую клавишу мыши), а также перемещаться по плоскости рисования;

*Скорость выполнения шагов алгоритма возможно регулировать с помощью кнопок «+» и «-», одно нажатие на одну из этих кнопок соответственно увеличивает или уменьшает скорость на 0.5 секунды (на панели, находящейся

между ранее упомянутыми кнопками будет выводиться текущая скорость алгоритма). Стандартная скорость алгоритма — 1.0 секунды.

1.2. Уточнение требований после сдачи прототипа.

К первой версии программы будет изменен формат представления результата работы алгоритма в текстовом поле: вместо матрицы смежности, будет выводиться список всех ребер, составляющих минимальное остовное дерево. Помимо этого, будет изменено представление матрицы смежности графа внутри файла: теперь, при вводе матрицы, пользователю обязательно будет требоваться указать названия вершин выше столбцов этой матрицы (опционально можно также указать названия вершин слева от строк), либо же пользователь сможет указать только нижнетреугольную матрицу.

На имена вершин налагаются следующие ограничения: имя должно состоять только из латинских букв и иметь длину не более трёх символов.

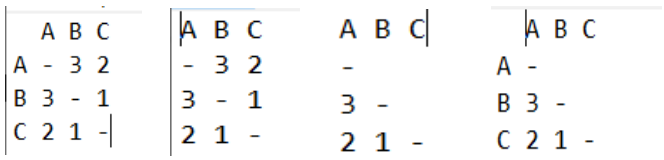


Рисунок 4 — Различные способы задания матрицы смежности в файле.

1.3. Уточнение требований после сдачи 1-ой версии

Во второй версии программы будет дополнительно добавлена кнопка «Заново», которая будет переводить граф в начальное состояние для повторного запуска алгоритма. Также будет немного подкорректирован интерфейс загрузки графа из файла: при повторной загрузке по умолчанию будет выбираться папка, из которой был взят файл в прошлый раз. Также кнопка «Сбросить» будет переименована в кнопку «Очистить холст».

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

а) «Прототип» должен включать в себя логику и минимальный графический интерфейс: холст, кнопки «Load graph» и «Start», первая — загружает граф из файла и отображает его на холсте, вторая — выводит минимальное остовное дерево графа в виде матрицы в текстовом поле, в текстовом поле на данном этапе будет реализован только вывод матрицы минимального остовного дерева.

*Прототип будет готов к 06.07.2022.

б) Первая версия должна включать весь функционал прототипа, однако кнопка «Start» получит новый функционал — помимо вывода матрицы в текстовое поле, она будет отображать на холсте минимальное остовное дерево графа. Будет добавлена кнопка «Reset», которая будет удалять граф с холста. Будет добавлена кнопка «Next step» - пошагового (по «маленьким» шагам) решения (в соответствии со спецификацией). На каждом шаге алгоритма вершины и рёбра будут раскрашиваться в соответствии со спецификацией.

*Первая версия будет готова к 08.07.2022

в) Вторая версия должна включать весь функционал первой версии, кнопки регулирующие скорость работы алгоритма, а также возможность задания графа с помощью мышки и холста (которая будет доступна через кнопку «Draw graph mode»), возможность перемещения вершин и камеры относительно холста (всего графа целиком), а также текстовое поле получит новый функционал: в нем будет выводиться информация обо всех «маленьких» и «больших» шагах. Также кнопка «Start» при нажатии будет меняться на кнопку «Stop», останавливающую текущую работу алгоритма. Кнопка «Stop» при нажатии будет меняться обратно на кнопку «Start». Также будет добавлена

регулировка скорости: кнопками “+” и “-”, а также с помощью текстового поля, в котором отображается текущая скорость. Кнопка «Reset» изменит название на «Clear canvas» и будет очищать холст (удалять граф). Будет добавлена кнопка «Again», которая будет перезапускать алгоритм (то есть отменять текущий прогресс алгоритма, как будто алгоритм и не запускался). При повторном выборе файла в окне выбора будет появляться папка, из которой был взят файл в прошлый раз.

*Вторая версия будет готова к 11.07.2022

2.2. Распределение ролей в бригаде

1) На логику будет выделено два человека: Корсунов Антон и Самулевич Василий: Антон будет разрабатывать структуру данных и интерфейс логики (для взаимодействия с графикой), Василий будет разрабатывать алгоритм.

2) На GUI будет выделен один человек — Сабанов Петр. Петр напишет графическое представление для прототипа, также в его обязанности в ходе дальнейшей разработки входит написание холста и возможностей взаимодействия с ним мышью.

*В дальнейшем (после разработки прототипа) к Петру присоединятся другие члены команды, в обязанности которых будет входить разработка структуры данных для графического отображения графа, а также связь графического представления с логикой.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Логика программы

Реализация логики программы содержится в 4 классах, расположенных в пакете com.leti.summer_practice.logic: Graph, Algorithm, Logic и LogicInterface. Далее будет представлено подробное описание каждого из них.

3.1.1. Graph

Этот класс используется в программе для представления соответствующей структуры данных.

Особенности реализации: внутри объекта данного класса для хранения графа используется матрица смежности (поле matrix). Также, поскольку все вершины в программе представляются в виде имен, внутри объекта есть хэш-таблица (поле address), ключи которой – имена вершин, а значения – номера соответствующих им строк (и столбцов) в матрице смежности. Еще класс Graph поддерживает операции добавления и удаления ребер и вершин. Поскольку с ребрами эти операции тривиальны (по именам начальной и конечной вершины нужно просто найти соответствующую ячейку и изменить в ней значение), подробное описание здесь будет представлено только для второго.

Добавление:

Для новой вершины получают номер пустой строки (и столбца) в матрице смежности, после чего связывают ее с этим номером с помощью поля address. Алгоритм получения номера: Сначала проверяется, остались ли в матрице смежности пустые строки и столбцы (их номера хранятся в списке avalible_numbers). Если остались, то из avalible_numbers удаляется последний элемент и возвращается в качестве результата. В противном случае происходит расширение матрицы смежности: с помощью метода resize_matrix в нее добавляется новая строка и столбец. После расширения в качестве результата возвращается номер соответствующий только что добавленным элементам матрицы.

Удаление:

По имени удаляемой вершины получается ее номер в матрице смежности, после чего этот номер добавляется в список avalible_numbers, а значения в соответствующей строке (и столбце) устанавливаются в null.

С целью создания более понятного внешнего интерфейса, а также для удобного использования в программе таких частей графа как ребро и вершина, внутри Graph было реализовано два вложенных класса: Node и Edge. Конструкторы обоих этих классов являются приватными, благодаря чему создавать соответствующие объекты можно только обращаясь к специальным методам Graph (например, create_vertex). Также для этих классов были переопределены методы equals и hashCode, благодаря чему их объекты корректно обрабатываются в хэш-таблицах.

Объекты типа Node имеют лишь одно поле – String name, в то время как Edge хранит начальную и конечную вершины, а также вес ребра (Node start, Node finish, int weight).

Внешний интерфейс класса Graph:

- Методы удаления и добавления: create_vertex, delete_vertex, create_edge, delete_edge.

  Особенности реализации этих методов уже были описаны ранее.

- Методы получения содержимого графа: edge_exist, vertex_exist, get_vertex, get_edge, get_vertices, get_edges.

  Первые два метода проверяют, содержится ли в графе соответствующее ребро или вершина: в качестве параметров передается одно (для вершины) или два (для ребра) имени, а результатом является true или false.

  get_vertex и get_edge имеют аналогичные наборы параметров, но отличаются возвращаемыми значениями: если соответствующая вершина

(ребро) существует, то возвращается построенный для нее объект типа Node(Edge), а если нет, бросается исключение.

get_vertices и get_edges возвращают ArrayList объектов соответствующего типа (Node или Edge).В случае всех вершин достаточно вернуть все ключи таблицы address, а в случае всех ребер происходит проход по всем занятым ячейкам матрицы смежности (номера всех занятых строк и столбцов хранятся в adress в качестве значений).

- Метод, реализующий считывание из файла: read_file.

  Этот статический метод получает на вход объект типа File и пытается считать из него граф. Если считывание прошло успешно, возвращается новый объект типа Graph, а если нет - бросается исключение.

  Особенности реализации: для создания графа по файлу был написан вложенный класс Graph_builder который работает по принципу паттерна строитель. Само создание происходит в несколько этапов: сначала вызывается метод init_names, в который передается первая строка файла (согласно спецификации она должна содержать имена всех вершин). Внутри этого метода в новый граф добавляются все вершины с перечисленными в строке именами. После этого идет построчное считывание с передачей полученной строки в метод read_row, который использует ее для инициализации длин ребер. После завершения считывания новый граф получается вызовом метода get_result.

- Методы, необходимые для реализации алгоритма Борувки: get_neighbors, is_connected

  get_neighbors получает на вход вершину (объект класса Node) и возвращает ArrayList вершин, связанных с переданной.

is_connected проверяет, является ли граф связным. Для этого внутри метода запускается обход в глубину, после чего проверяется, были ли посещены все вершины: если да, то граф связный, а если нет – несвязный.

- Вспомогательные методы, необходимые для графического интерфейса: clear, get_vertex_count.

  Первый метод делает граф пустым, а второй – возвращает количество вершин.

3.1.2. Algorithm

Этот класс отвечает за сам алгоритм Борувки, для этого он использует Graph для работы с переданным пользователем графом.

Особенности реализации: при создании объекта данного класса в конструктор передается объект класса Graph. В конструкторе класса Algorithm объекту передаются следующие составляющие графа:

1) Список вершин — записывается в поле vertices (ArrayList вершин) — служит для более простой работы с вершинами графа(более подробное использование описано в методах класса);

2) Список ребер — записывается в поле edges (ArrayList ребер) — служит для более простой работы с ребрами графа (более подробное использование описано в методах класса);

3) Число вершин — записывается в поле length_components (int) – служит для хранения числа компонент на каждом шаге алгоритма;

К тому же, в конструкторе графа выделяется память под следующие поля:

1) temporary_graph (Grapg) — служит для хранения минимального остовного дерева графа, изначально состоит только из всех переданных вершин, но на каждом шаге алгоритма к нему добавляются ребра, принадлежащие минимальному остовному дереву;

2) edges_color (Map<Edge, Integer>) — хэш-таблица, ключами которой являются ребра графа, а значения числа типа Integer, определяющие принадлежность ребер к соответствующим компонентам связности, изначально это поле является пустым;

3) hashTableNode (Map<Edge, Integer>) — хэш-таблица, ключами которой являются вершины графа, а значения числа типа Integer, определяющие принадлежность вершин к соответствующим компонентам связности, изначально значения у всех ключей (т. е. вершин графа) разные значения, т. к. в начале работы алгоритма каждая вершина образует отдельную компоненту связности;

Основные методы класса:

1) get_new_edges – возвращает массив ребер, которые войдут в компоненты связности (а позже — в минимальное остовное дерево), в конце шага алгоритма, и добавляет эти ребра в графу temporary_graph. Метод перебирает все ребра графа в цикле, на каждой итерации ищет минимальное ребро для каждой компоненты связности (при условии, что это ребро соединяет вершины, относящиеся к разным компонентам);

2) next_step — в случае, если компонент связности больше одной, вызывает метод update_components, чтобы перейти к концу «большого» шага;

3) update_components – соединяет компоненты связности, у которых появился путь к друг другу после вызова метода get_new_edges и возвращает новое количество компонент в графе temporary_graph. Метод создает «открытый» список вершин (в начале которого входят все вершины), просматривает каждую вершину из списка и запускает обход в глубину для нее и для всех вершин, к которым она имеет путь (метод dfs), в ходе обхода все вершины и ребра, соединяющие их, «сливаются» в одну компоненту.

4) isFinished – проверяет, закончен ли алгоритм или нет;

5) get_answer – в случае, если алгоритм закончен, возвращает все ребра минимального остовного дерева;

6) get_vertex_color – принимает вершину и возвращает номер, отвечающий за ее цвет;

7) get_edge_color – принимает две вершины и, в случае если между ними существует ребро, возвращает номер, отвечающий за его цвет.

3.1.3. Logic и LogicInterface

LogicInterface представляет из себя интерфейс, с помощью которого графическая часть программы взаимодействует с логикой. В свою очередь класс Logic этот интерфейс реализует. Поскольку набор методов LogicInterface и Logic полностью совпадают, здесь будет рассмотрен только последний. Единственное, о чем стоит упомянуть в LogicInterface, так это о двух вложенных статических классах NodeInfo и EdgeInfo: оба они были созданы для более удобной передачи графическому интерфейсу запрашиваемых им данных. Объекты типа NodeInfo содержат два поля – имя и номер цвета вершины (String name, Integer color). В свою очередь EdgeInfo хранит имена начальных и конечных вершин, вес и цвет ребра (String start, String finish, int weight, Integer color).

Особенности реализации Logic: этот класс имеет два поля – graph и algorithm. Первое из них хранит граф, который видит пользователь на экране, а второе инкапсулирует в себе все данные, касающиеся алгоритма Борувки, а именно информацию о том какая вершина (или ребро) какой компоненте связности принадлежит (в дальнейшем описании цвет вершины и номер компоненты связности, которая эту вершину содержит, будут иметь одинаковый смысл). В конструкторе Logic поле graph инициализируется пустым графом, а algorithm остается равным null: для его инициализации графическому интерфейсу необходимо вызвать метод start_algorithm.

Поскольку GUI отображает работу алгоритма Борувки пошагово, в логике он также работает по шагам. Для перехода между ними был написаны методы getNewEdge и nextBigStep. С целью объяснения того, что они делают, формализуем понятие шага: во время его выполнения к каждой компоненте связности добавляется минимальное инцидентное ей ребро, после чего происходит обновление этих компонент. Непосредственно сам шаг алгоритма выполняется в nextBigStep, а getNewEdge последовательно (по одной штуке за вызов) возвращает ребра графа, которые будут добавлены на следующем шаге.

Внешний интерфейс класса Logic:

- Методы изменения графа: removeVertex, addVertex, addEdge, removeEdge, clearGraph,loadFile

  Их реализация сводится к вызову соответствующих методов класса Graph.

- Методы получения содержимого графа: vertexExist, edgeExist, getVertexInfo, getEdgeInfo, getVetrices, getEdges

  vertexExist и EdgeExist проверяют, содержится ли в графе переданное ребро/вершина.

  Оставшиеся 4 метода повторяют функционал соотв. методов класса Graph за исключением того, что для каждой вершины и ребра они указывают ее цвет (который они получают с помощью вызовов get_vertex_color и get_edge_color класса Algorithm).

- Методы, созданные для управления работой алгоритма: startAlorithm, getNewEdges, nextBigStep, isAlorithmStarted, isAlorithmFinished, getAnswer, killAlogrithm

  Первые 3 метода уже были описаны ранее.

isAlorithmStarted и isAlorithmFinished проверяют, был ли алгоритм запущен (поле algorithm не равно null) /завершен (метод isFinished возвращает true), и возвращают соответствующее значение типа boolean.

getAnswer возвращает Arraylist ребер, которые составляют минимальное остовое дерево. Этот метод работает, только если алгоритм был уже завершен (в противном случае бросается исключение).

killAlgorithm прерывает выполнения алгоритма, устанавливая в поле algorithm значение null.

3.2. GUI

3.2.1. Холст

Для отрисовки векторной графики был создан класс VectorCanvas.

Этот класс является обёрткой для другого класса — VectorCanvasContent, на котором как раз и отображаются все рисуемые элементы.

Такая реализация была выбрана для того, чтобы можно было перемещать холст, не перемещая экземпляр класса VectorCanvas (так как на самом деле перемещается VectorCanvasContent, а VectorCanvas лишь реагирует на события, генерируемые пользователем).

Основные методы VectorCanvas:

1) void clear() — очищает холст (очищает VectorCanvasContent);
2) void draw() — отрисовывает холст (рисователь задаётся отдельно);
3) void redraw() — сначала очищает, а потом отрисовывает холст;
4) Consumer<VectorCanvas> getDrawer() — возвращает рисователь;

5) void setDrawer(Consumer<VectorCanvas> drawer) — устанавливает рисователь;
6) VectorCanvasContent getContent() — возвращает экземпляр класса VectorCanvasContent;
7) void draw(Shape shape) — отрисовывает фигуру на холсте;
8) void erase(Shape shape) — удаляет фигуру с холста.

Для непосредственной отрисовки графа был создан класс GraphCanvas, который наследуется от VectorCanvas.

Для отрисовки и манипуляций с графическим представлением графа он содержит следующие поля:
1) Map<String,Circle> verticesMap — ассоциативный массив для доступа к кругам (графически представляющим вершины) по имени;
2) Map<Circle,String> reversedVerticesMap — ассоциативный массив, содержащий имена по кругам;
3) Map<String,Text> verticesTextsMap — ассоциативный массив, содержащий тексты (графическое представление имён вершин) по именам вершин;
4) Map<Pair<String,String>,Line> edgesMap — ассоциативный массив, содержащий линии (графическое представление рёбер графа) по парам (начало, конец);
5) Map<Line,Pair<String,String>> reversedEdgesMap — ассоциативный массив, содержащий пары (начало, конец) по линиям;
6) Map<Pair<String,String>,Text> edgesTextsMap — ассоциативный массив, содержащий тексты (веса рёбер) по парам (начало, конец);
7) Set<LogicInterface.EdgeInfo> specialColorEdges — множество рёбер, которые должны быть отрисованы с особым цветом на текущем шаге.

GraphCanvas содержит следующие основные методы:

1) Color getColorByInt(int n) — получает цвет вершины или ребра по номеру, означающему цвет, взятому из логики;

2) void initializeWithLogic(LogicInterface logic) — добавляет в ассоциативные массивы информацию о вершинах и рёбрах из логики;

3) void notifyColorsChanged() — информирует GraphCanvas о том, что цвет некоторых рёбер и/или вершин сменился. В этом методе устанавливаются цвета для всех кругов и линий;

4) GraphMode getGraphMode(), void setGraphMode(GraphMode graphMode) — геттер и сеттер для режима графа (режим рисования или режим перетаскивания).

Для обработки пользовательских событий были написаны два класса: CircleEvents и LineEvents. Их экземпляры создаются внутри экземпляра GraphCanvas, а обработчики событий, хранящиеся в этих классах, устанавливаются кругам и линиям.

При совершении следующих действий: создание вершины и создание ребра — появляется диалоговое окно, в котором пользователя просят ввести имя новой вершины или вес нового ребра соответственно.

3.2.2. Создание представления

Представление задаётся в файле summer-practice.fxml и загружается в классе SummerPracticeApplication.

3.2.3. Таймер

Для корректной работы автоматического применения алгоритма необходим таймер. Для этой задачи был написан класс SingleTaskTimer.

Этот таймер по умолчанию имеет всего одно задание. При попытке начать новое задание, старое отменяется.

При каждом успешном выполнении задания значение времени его выполнения сохраняется в таймере, чтобы потом его можно было использовать для запуска нового задания, если ему будет необходима определённая задержка со времени успешного выполнения предыдущего задания.

3.2.4. Контроллер

Как указано в файле summer-practice.fxml, контроллером представления является SummerPracticeController.

Поля класса SummerPracticeController:

1) LogicInterface logic — экземпляр класса Logic, ответственный за всю логику алгоритма и представления графа;
2) boolean answerAlreadyPrinted — булеан-переменная, хранящая информацию о том, был ли уже выведен ответ;
3) SingleTaskTimer autoStepTimer — таймер, необходимый для автоматического выполнения алгоритма.

Основные методы класса SummerPracticeController:

1) void initialize(URL location, ResourceBundle resources) — метод для инициализации объекта класса. Вызывается автоматически системой JavaFX. В этом методе проводятся начальные действия и настройка различных компонентов вида;

2) void onClearGraphClicked(ActionEvent actionEvent) — метод, вызываемый при нажатии на кнопку «Очистить граф». При нажатии на кнопку появляется окно подтверждения, в котором пользователь должен подтвердить, что он хочет очистить граф. Если последовало подтверждение, холст и текстовое поле для логов очищаются, также очищается логическое представление графа;
3) void onLoadFromFileClick(ActionEvent actionEvent) — при нажатии на кнопку «Загрузить из файла» появляется стандартный для системы диспетчер файлов, в котором можно выбрать файл. Если загрузка файла потерпела неудачу, пользователю покажут сообщение об ошибке;
4) void onStartClick(ActionEvent actionEvent) — этот метод запускает таймер с заданием по вызову метода nextStep, если таймер ещё не запущен, и останавливает его, если он запущен. Также в методе меняется текст кнопки со «Старт» на «Стоп» и наоборот;
5) void onNextStepClicked(ActionEvent actionEvent) — метод, который должен переходить на новый шаг алгоритма. Если таймер для автоматического решения запущен, то он останавливается, затем совершается новый шаг (вызывается метод nextStep) алгоритма, затем таймер снова запускается с нужной задержкой. Если таймер не был запущен, то просто вызывается метод nextStep;
6) void nextStep() — метод, который переводит алгоритм на один шаг вперёд. В нём вызывается метод перерисовки холста, а также выводятся все сообщения в текстовое поле для логов;
7) void onAgainClicked(ActionEvent actionEvent) — в этом методе работа алгоритма прекращается. Если таймер был зарущен, он останавливается;
8) void onModeClick(ActionEvent actionEvent) — в этом методе меняется режим холста (режим рисования или режим перетаскивания). Также в этом методе изменяется текст кнопки с «Режим рисования» на «Режим перетаскивания» и наоборот;

9) void onMinusSpeedClick(ActionEvent actionEvent), void onPlusSpeedClick(ActionEvent actionEvent) — эти методы меняют скорость работы таймера.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование логики программы

4.1.1. Тестирование методов класса Graph

1) Тестирование считывания: read_file – считывает граф из файла, проверяя данные на корректность, и get_edges – возвращает список всех ребер графа

Таблица 1 — тестирование read_file и get_edges

№	Содержание файла	Ребра считанного графа	Комментарий
1	1	Ошибка (некорректные данные)	Тест пройден
2	A B C D E F - 1 - 3 - - 2 - 1 - - - 4 2 - - - - 8 - -	A --- B = 1 A --- D = 2 C --- D = 1 D --- E = 2 D --- F = 8 A --- C = 3 C --- E = 4	Тест пройден
3	A B C D - 1 - 4 1 - 2 - - 2 - 1 4 - 1 -	A --- B = 1 B --- C = 2 C --- D = 1 A --- D = 4	Тест пройден
4	A B C A - 1 2 B 1 - 1 C 2 1 -	A --- B = 1 B --- C = 1 A --- C = 2	Тест пройден

2) Тестирование добавления и удаления вершин и ребер:

Таблица 2 — тестирование create_vertex и delete_vertex

№	Тестируемый метод	Входные данные (имя вершины)	Вершины графа до вызова метода	Вершины графа после вызова метода	Комментарий
1	create_vertex	S	A, B, C	A, B, C, S	Тест пройден
2	create_vertex	A	A, C	Ошибка (вершина с таким именем уже существует)	Тест пройден
3	delete_vertex	S	A, B, C, S	A, B, C	Тест пройден
4	delete_vertex	S	A, C	Ошибка (такой вершины не существует)	Тест пройден

Таблица 3 — тестирование create_edge и delete_edge

№	Тестируемый метод	Входные данные	Ребра графа до вызова метода	Ребра графа после вызова метода	Комментарий
1	create_edge	A, D, 1	A --- B = 1 A --- C = 2 B --- C = 1 D --- B = 2	A --- B = 1 A --- C = 2 B --- C = 1 D --- B = 2 A --- D = 1	Тест пройден
2	create_edge	A, B, 1	A --- B = 1 A --- C = 2 B --- C = 1 D --- B = 2	Ошибка (такое ребро уже есть)	Тест пройден
3	delete_edge	A, D	A --- B = 1 A --- C = 2 B --- C = 1 D --- B = 2 A --- D = 1	A --- B = 1 A --- C = 2 B --- C = 1 D --- B = 2	Тест пройден
4	delete_edge	A, D	A --- B = 1 A --- C = 2	Ошибка (такого ребра не существует)	Тест пройден

			B --- C = 1 D --- B = 2		

3) Тестирование метода, проверяющего, является ли граф связным

Таблица 4 — тестирование is_connected

№	Вершины графа	Ребра графа	Результат работы	Комментарий
1	A, B, C, D	A --- B = 1 A --- C = 2 B --- C = 1 D --- B = 2 A --- D = 1	true	Тест пройден
2	A, B, C	A --- B = 1 B --- C = 1 A --- C = 2	true	Тест пройден
3	A, B		false	Тест пройден
4	A, B, C, D	A --- B = 1 C --- D = 2	false	Тест пройден

4.1.2 Тестирование методов класса Algorithm

Для наглядности тестирования был задействован интерфейс GUI.

1) Тестирование метода next_step

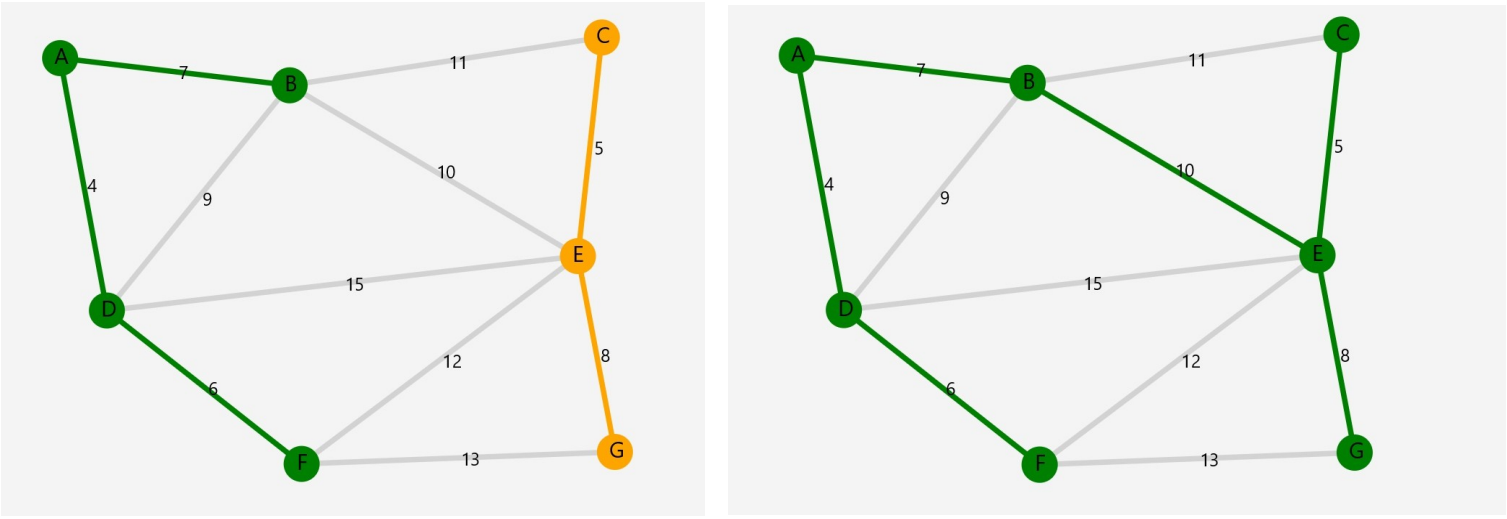


Рисунок 5 — Тест №1: компоненты связности до и после вызова метода

Комментарий: метод отработал корректно.

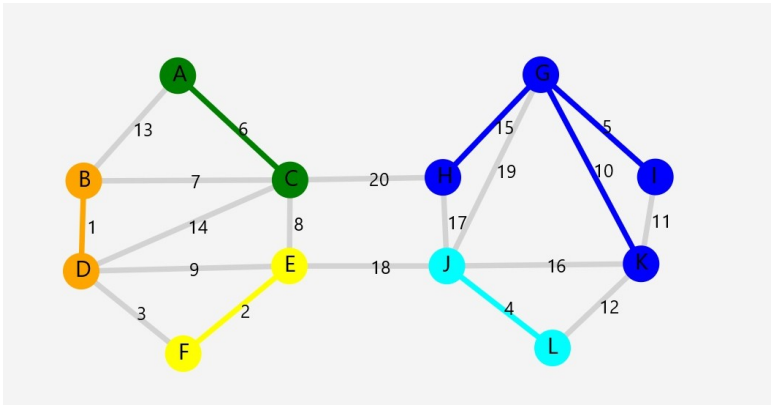


Рисунок 6 — Тест №2: компоненты связности до и после вызова метода

Комментарий: метод отработал корректно.

2) Тестирование метода get_new_edges

Тест №1:

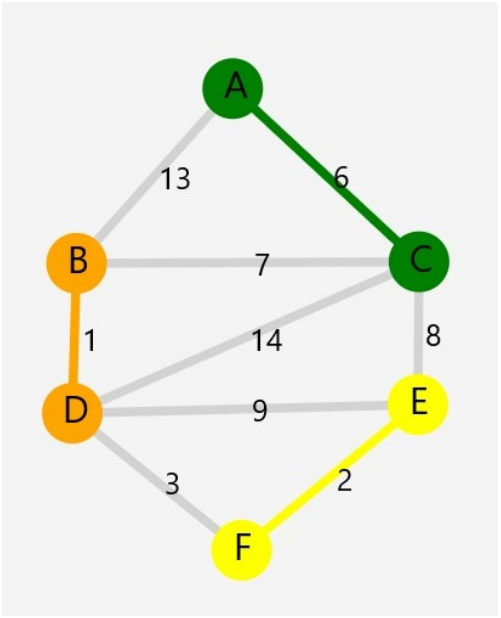


Рисунок 6 – состояние алгоритма до вызова метода.

Возвращаемый результат: BC, DF.

Комментарий: метод отработал корректно.

Тест №2:

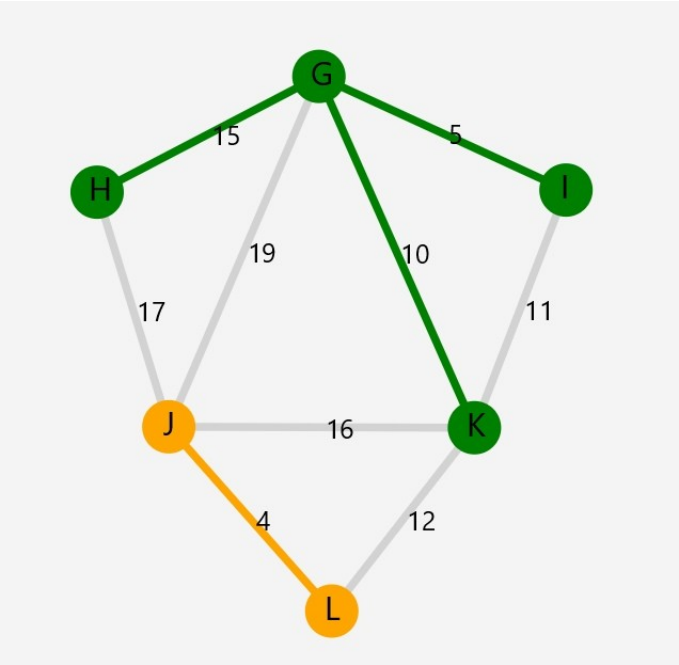


Рисунок 7 – состояние алгоритма до вызова метода.

Возвращаемый результат: LK.

Комментарий: метод отработал корректно.

3) Тестирование метода get_answer.

Таблица 5 – тестирование get_answer.

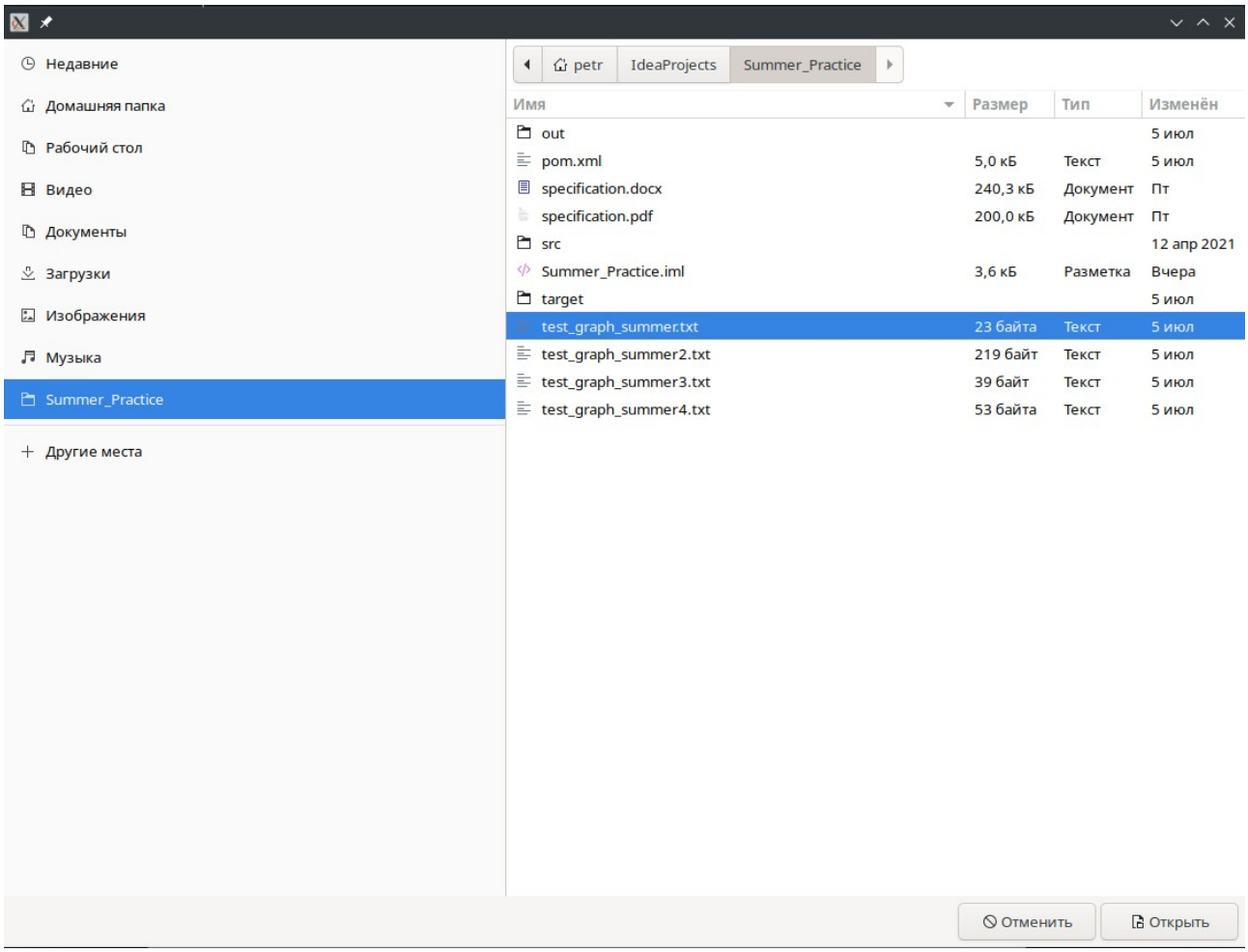
№	Ребра графа	Возвращаемый результат	Комментарий
1	A --- T = 1 A --- H = 34 T --- G = 3 T --- F = 564 G --- H = 5 G --- F = 7 H ---F = 234	A --- T = 1 T --- G = 3 G --- F = 7 G --- H = 5	Тест пройден

2	K --- T = 5 K --- A = 2 A --- T = 1	K --- A = 2 A --- T = 1	Тест пройден
3	K --- T = 66 K --- A = 2 T --- D = 6 D --- A = 35	K --- A = 2 D --- A = 35 T --- D = 6	Тест пройден

4.2. Тестирование графического интерфейса

4.2.1. Тестирование работы кнопок и текстовых полей

1. Кнопка «Загрузить из файла».



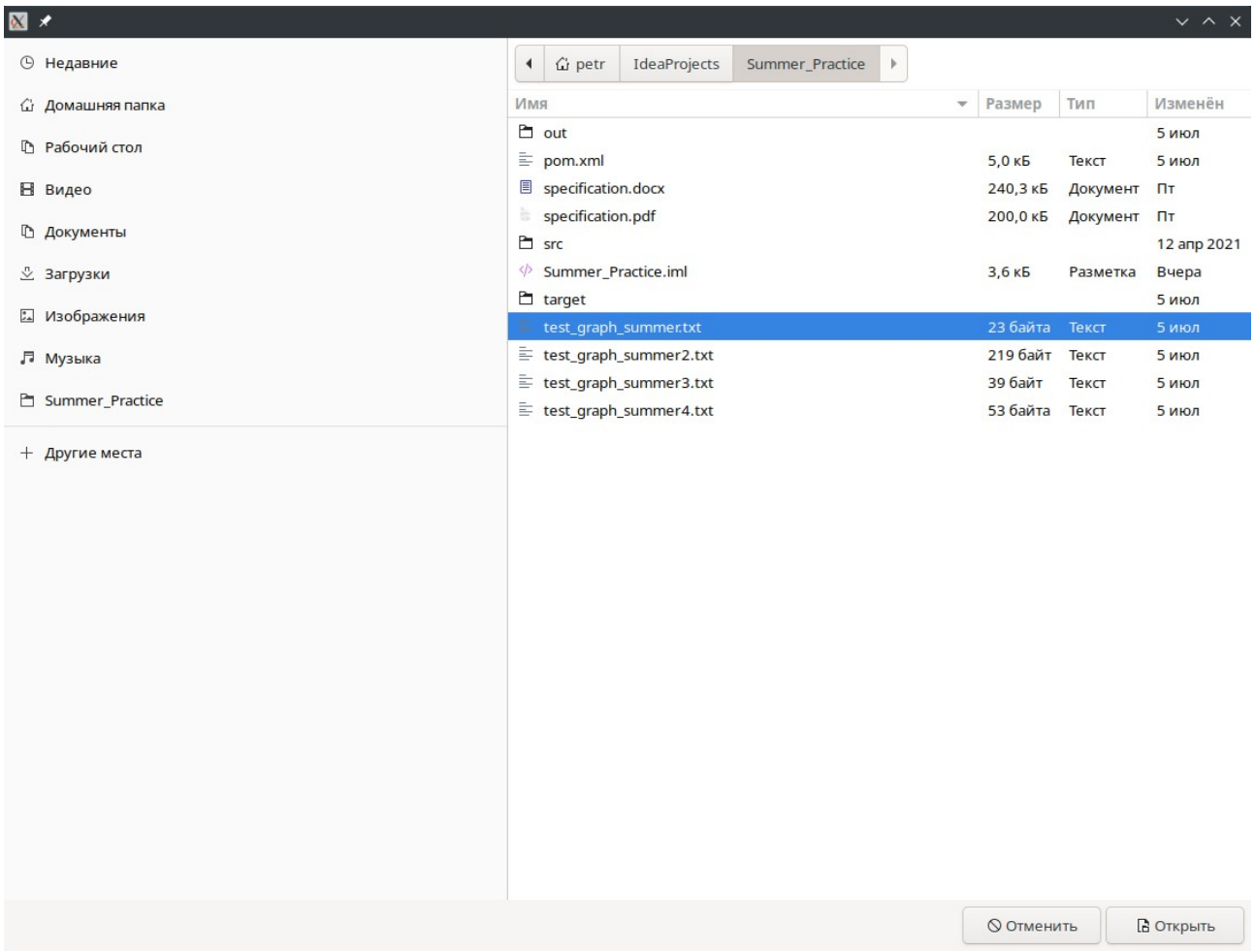


Рисунок 8 — окно выбора файла

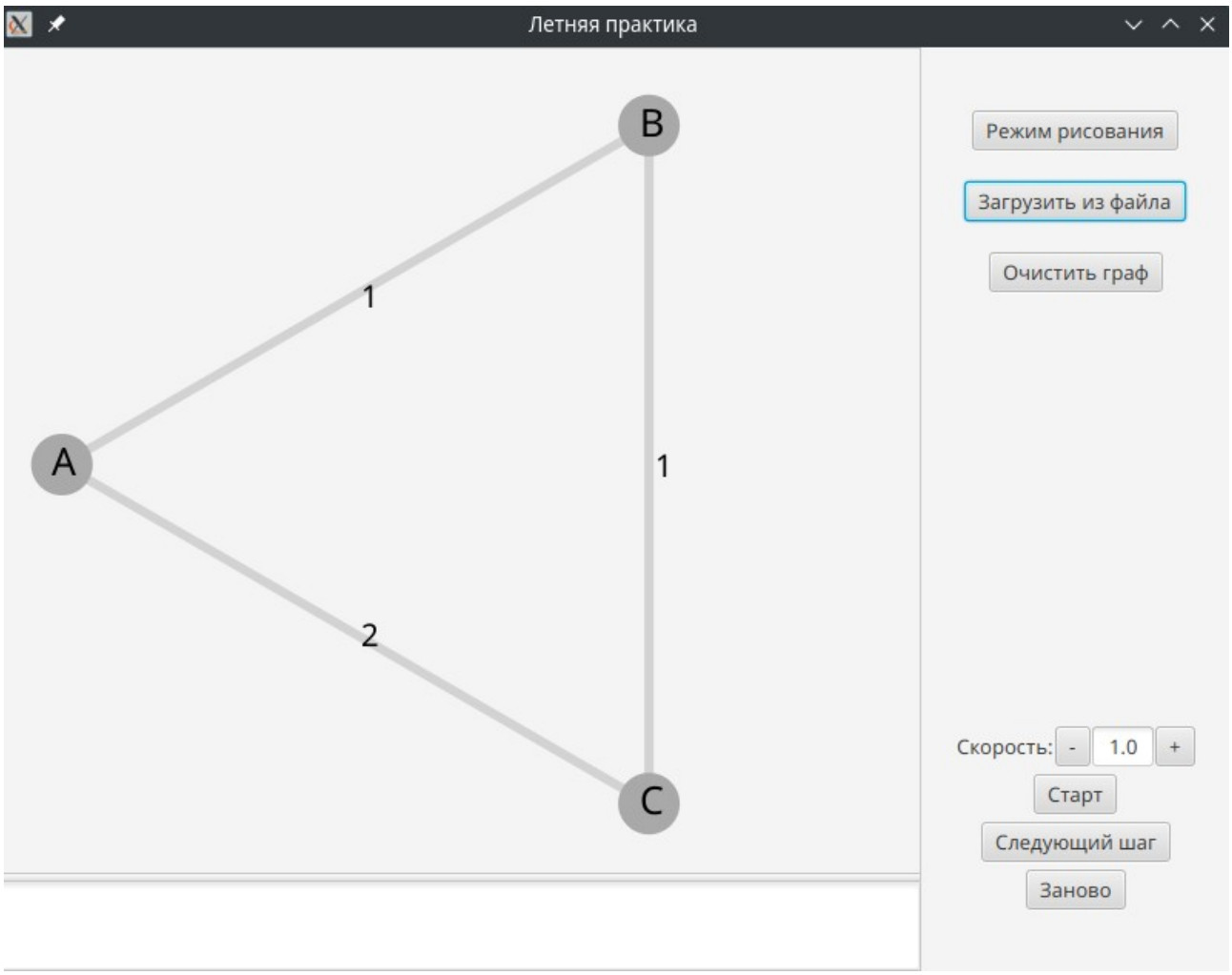


Рисунок 9 — успешная загрузка файла

Загрузка прошла успешно.

2. Кнопка «Очистить граф».

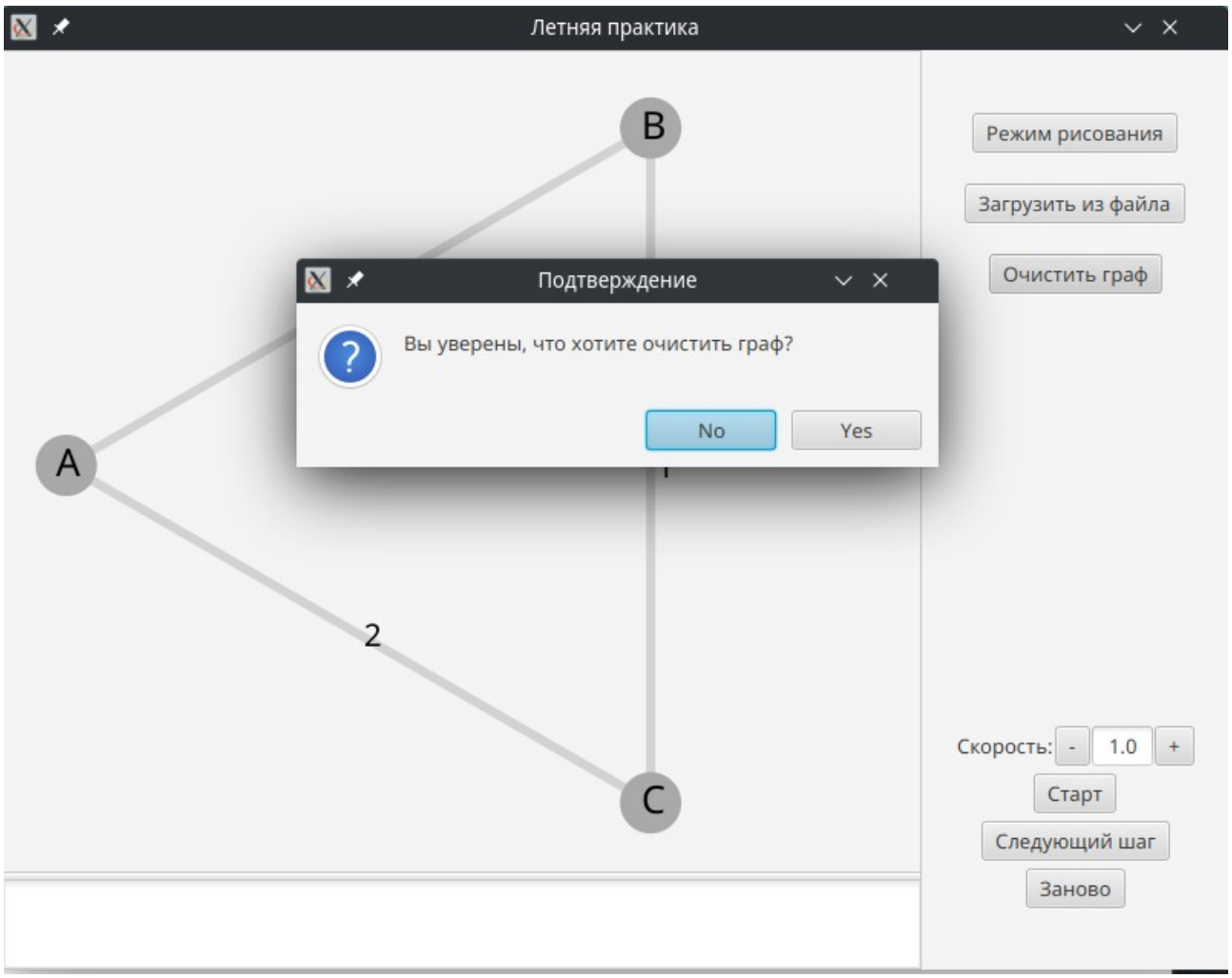


Рисунок 10 — окно с подтверждением очистки графа

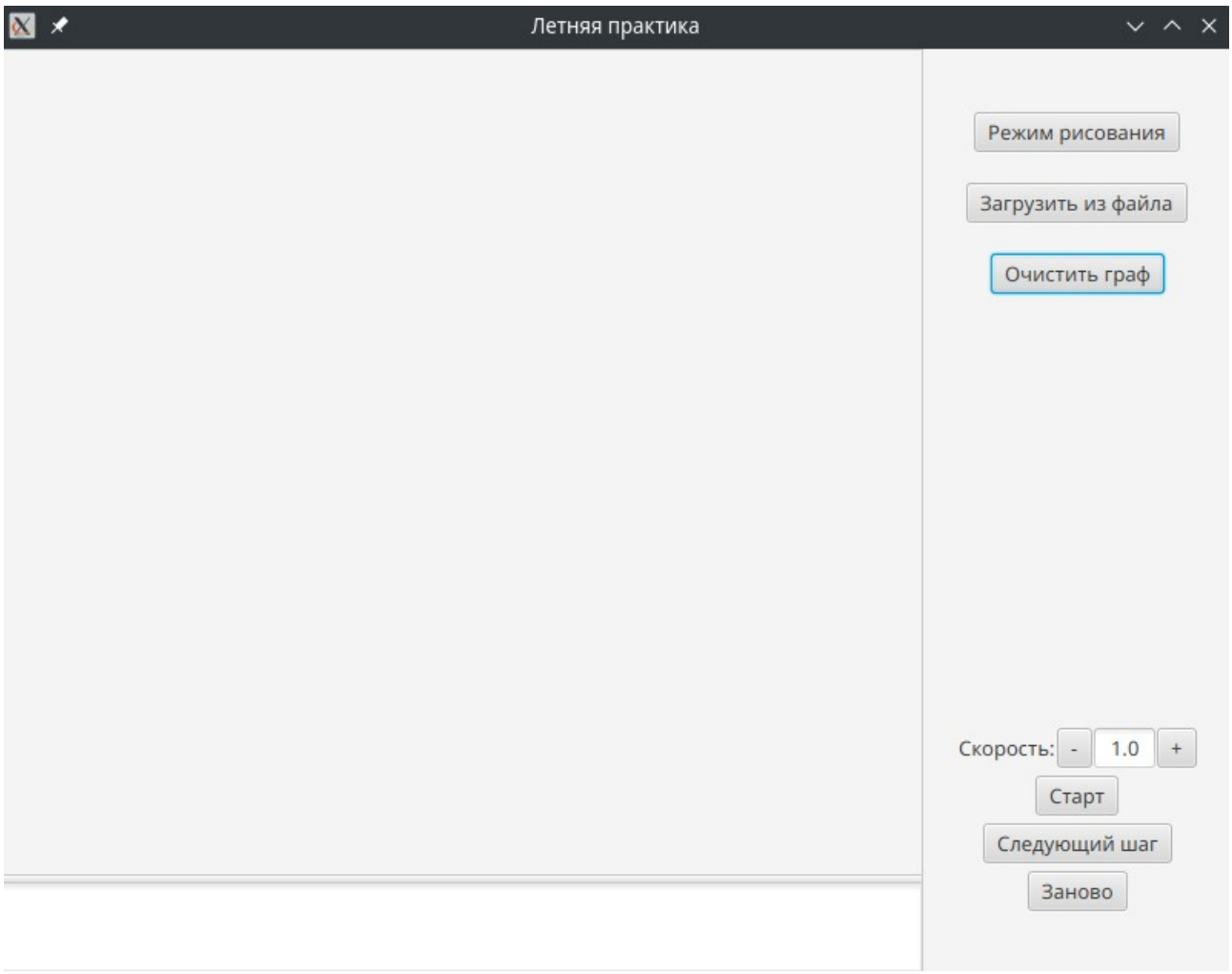


Рисунок 11 — очистка прошла успешно

Очистка прошла успешно.

3. Кнопка «Старт» («Стоп»), «Следующий шаг» и «Заново», а также текстового поля для логирования.

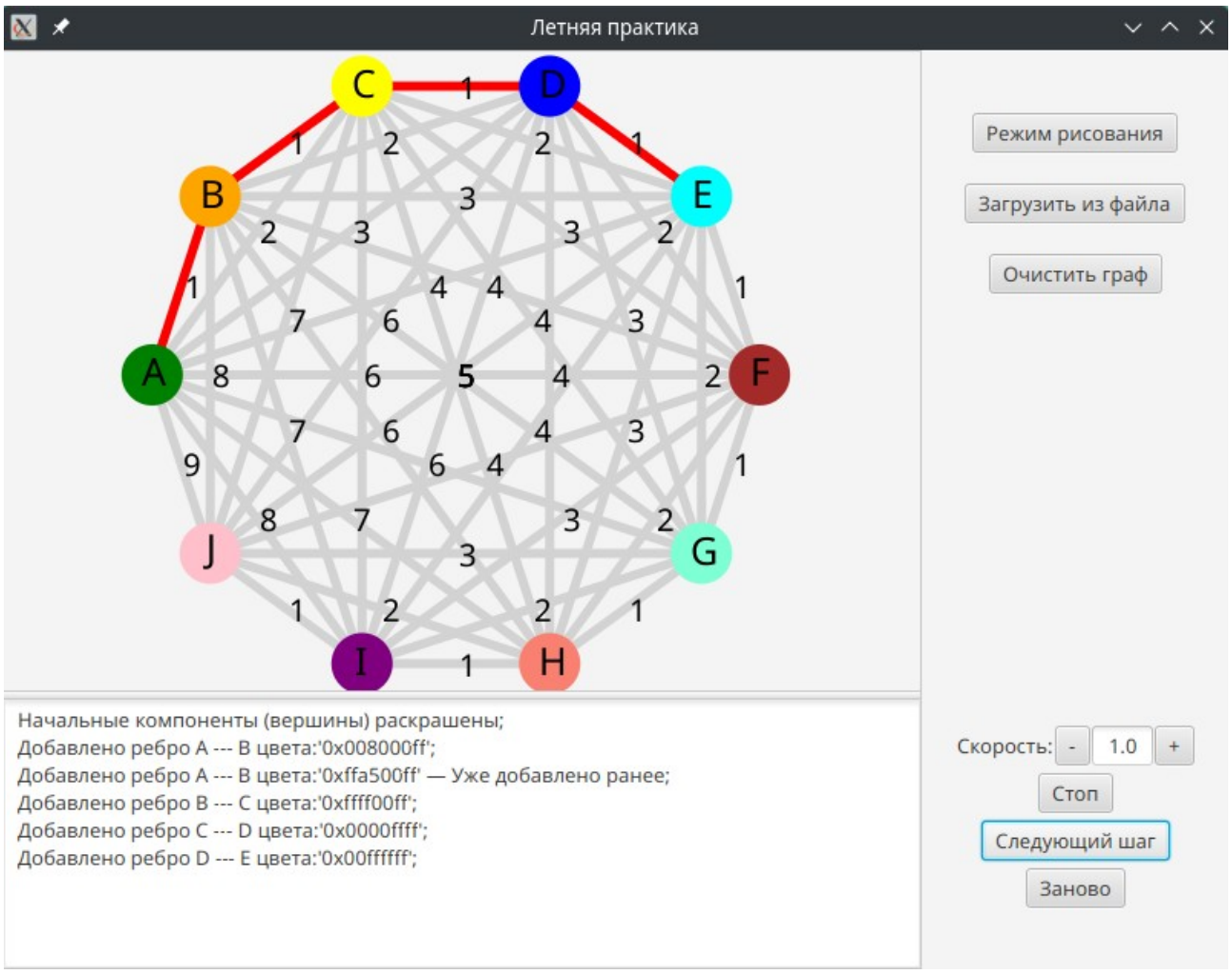


Рисунок 12 — выполнение алгоритма в действии

При нажатии на кнопку «Следующий шаг» выполняется следующий шаг алгоритма. При нажатии на кнопку «Старт», её текст меняется на «Стоп», а алгоритм начинает выполняться автоматически.

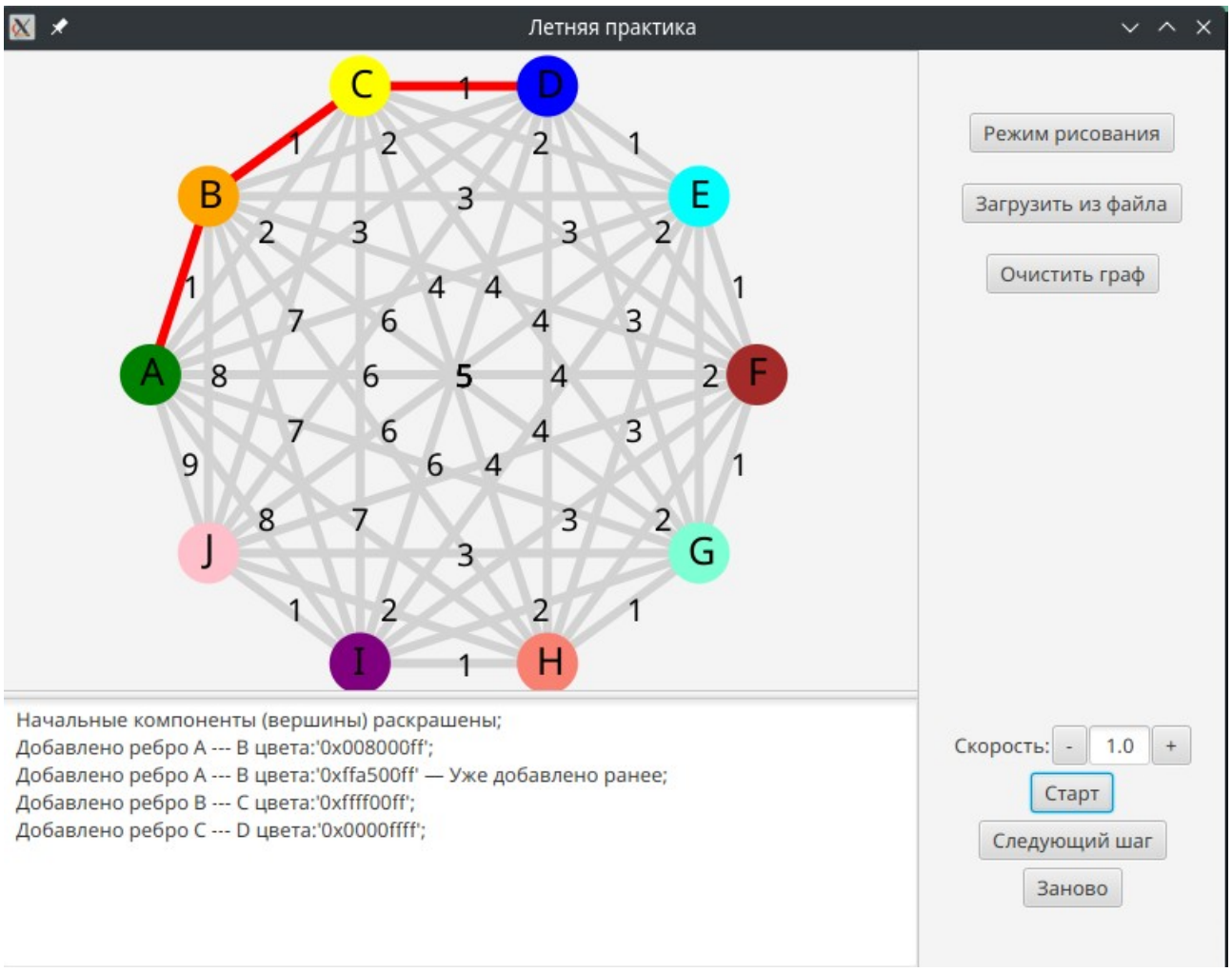


Рисунок 13 — после нажатия на кнопку «Стоп»

При нажатии на кнопку «Стоп», автоматическое выполнение алгоритма прекращается, а текст кнопки меняется на «Старт».

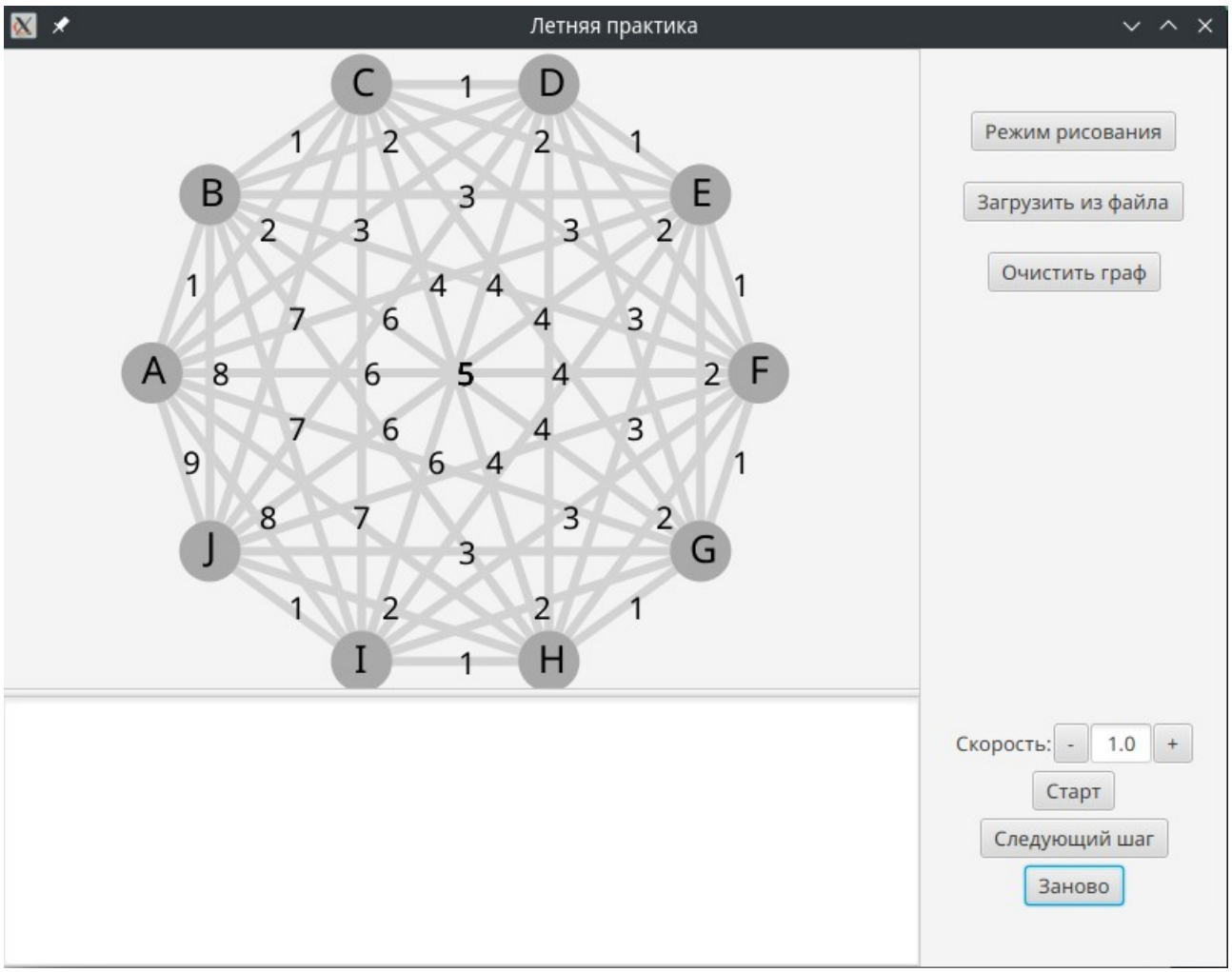


Рисунок 14 — после нажатия на кнопку «Заново»

При нажатии на кнопку «Заново» алгоритм останавливается, весь прогресс алгоритма сбрасывается.

Как видно из скриншотов, в текстовом поле для логирования отображается текущий прогресс алгоритма.

4.2.2 Тестирование возможностей холста

1. Создание новой вершины.

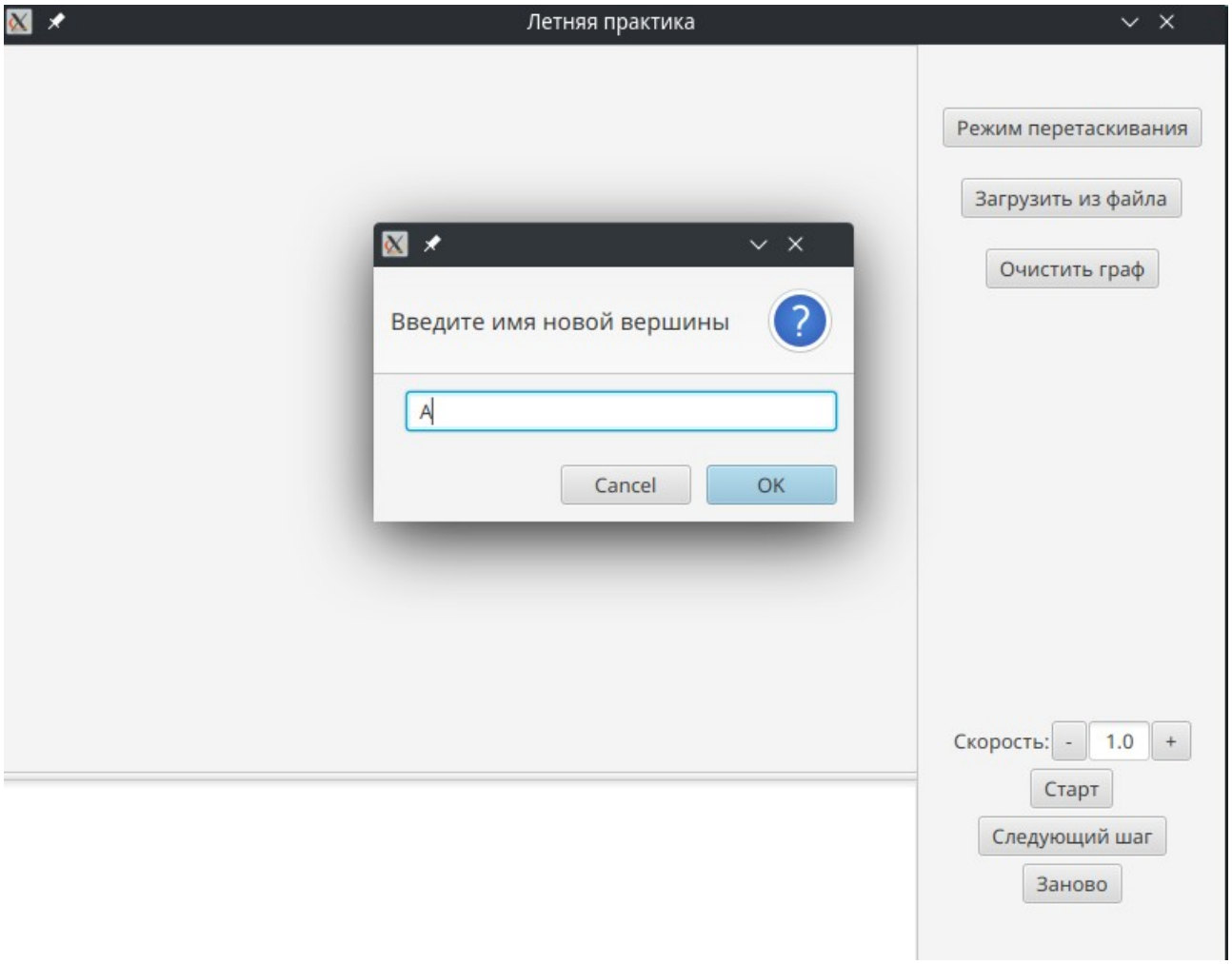


Рисунок 15 — ввод имени новой вершины

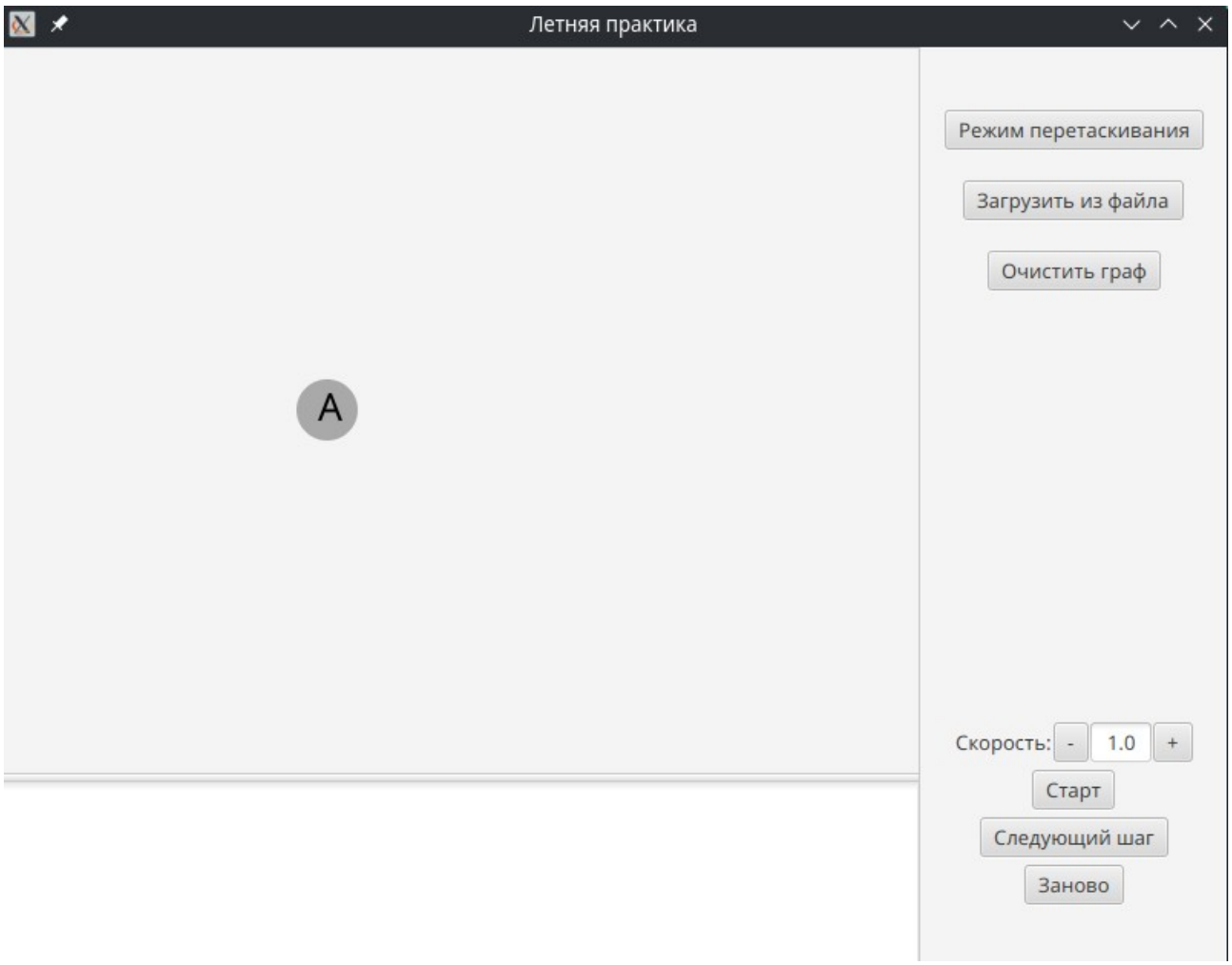


Рисунок 16 — новая вершина создана

Создание новой вершины работает нормально.

2. Создание нового ребра.

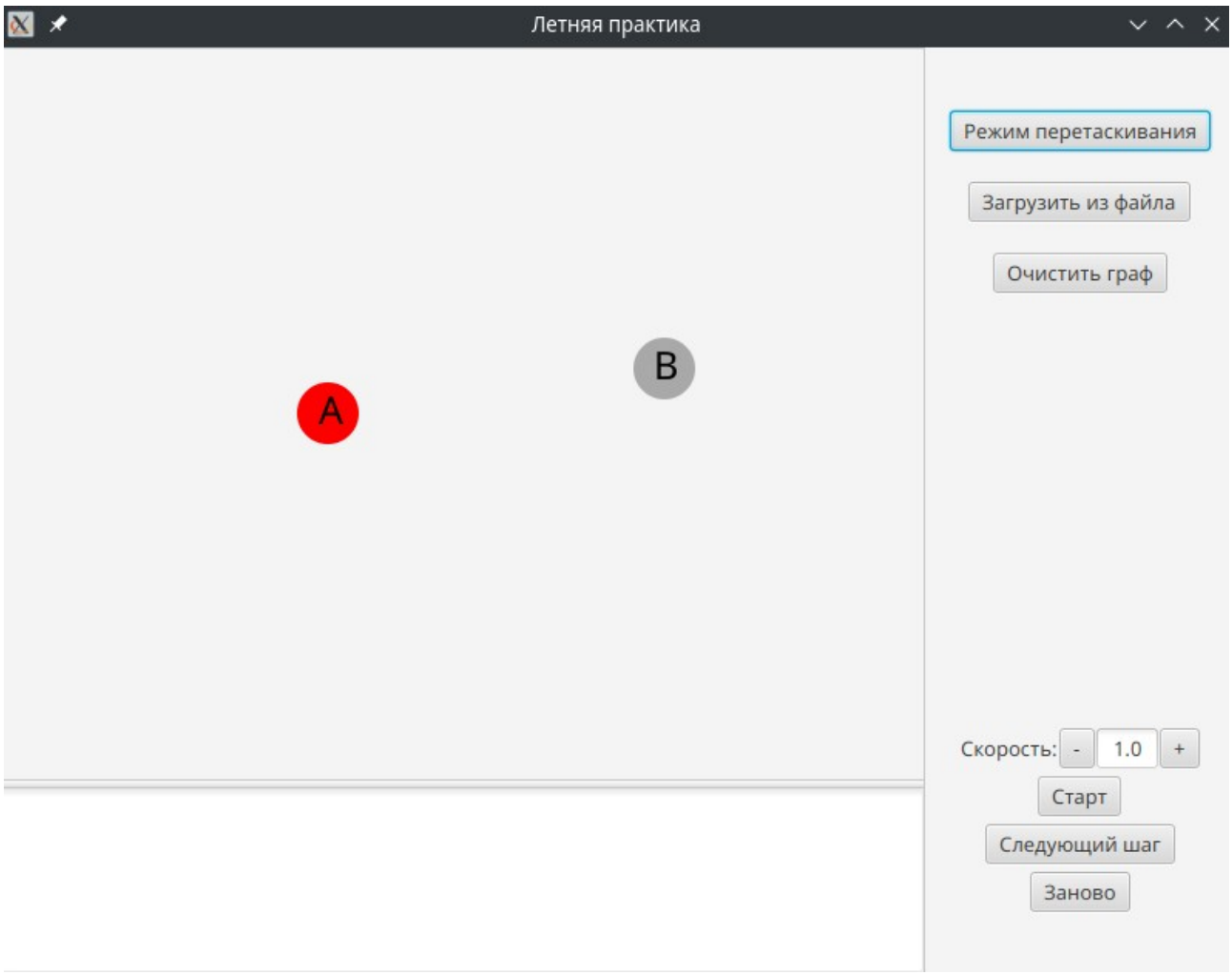


Рисунок 17 — выделение начальной вершины

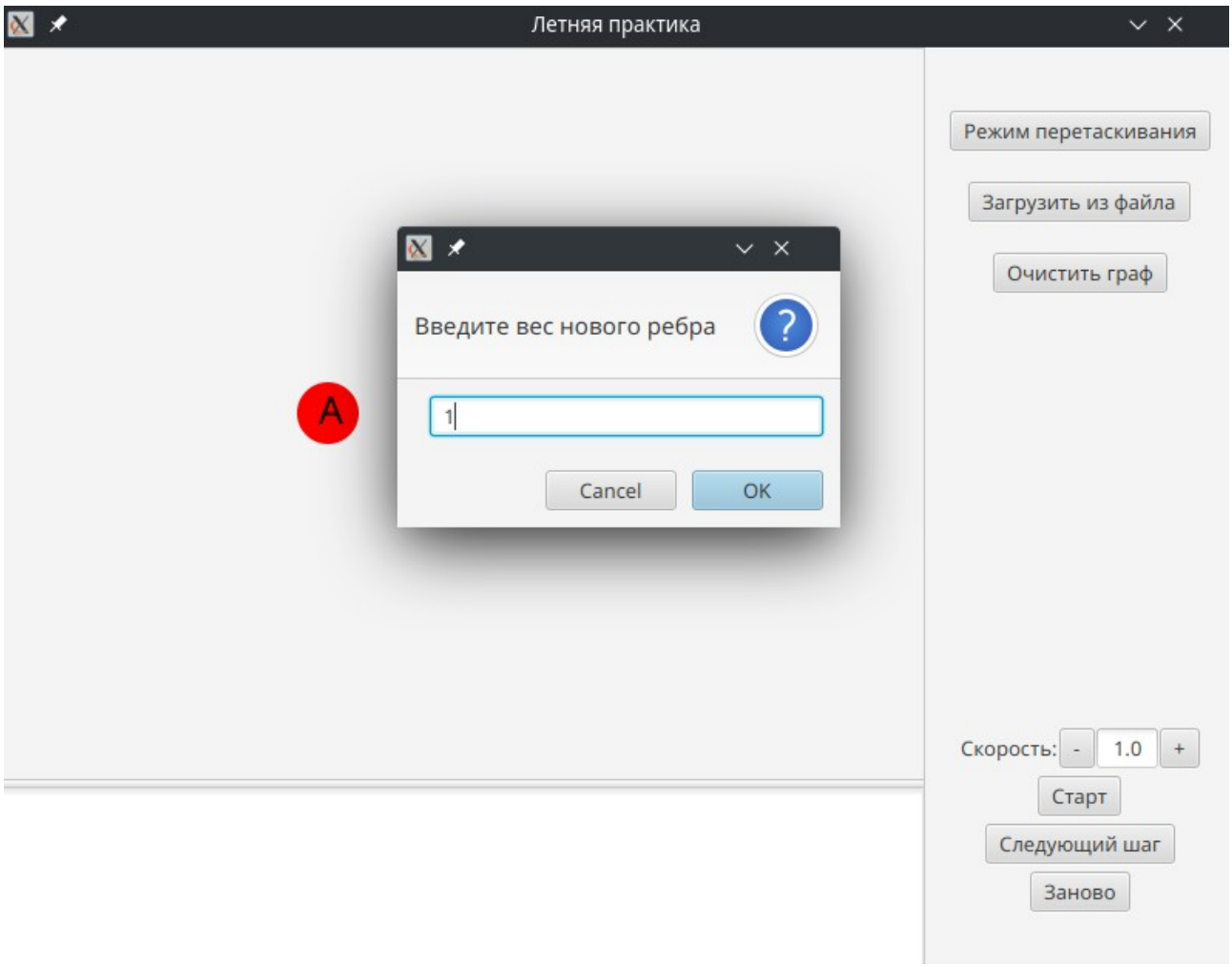


Рисунок 18 — ввод веса нового ребра

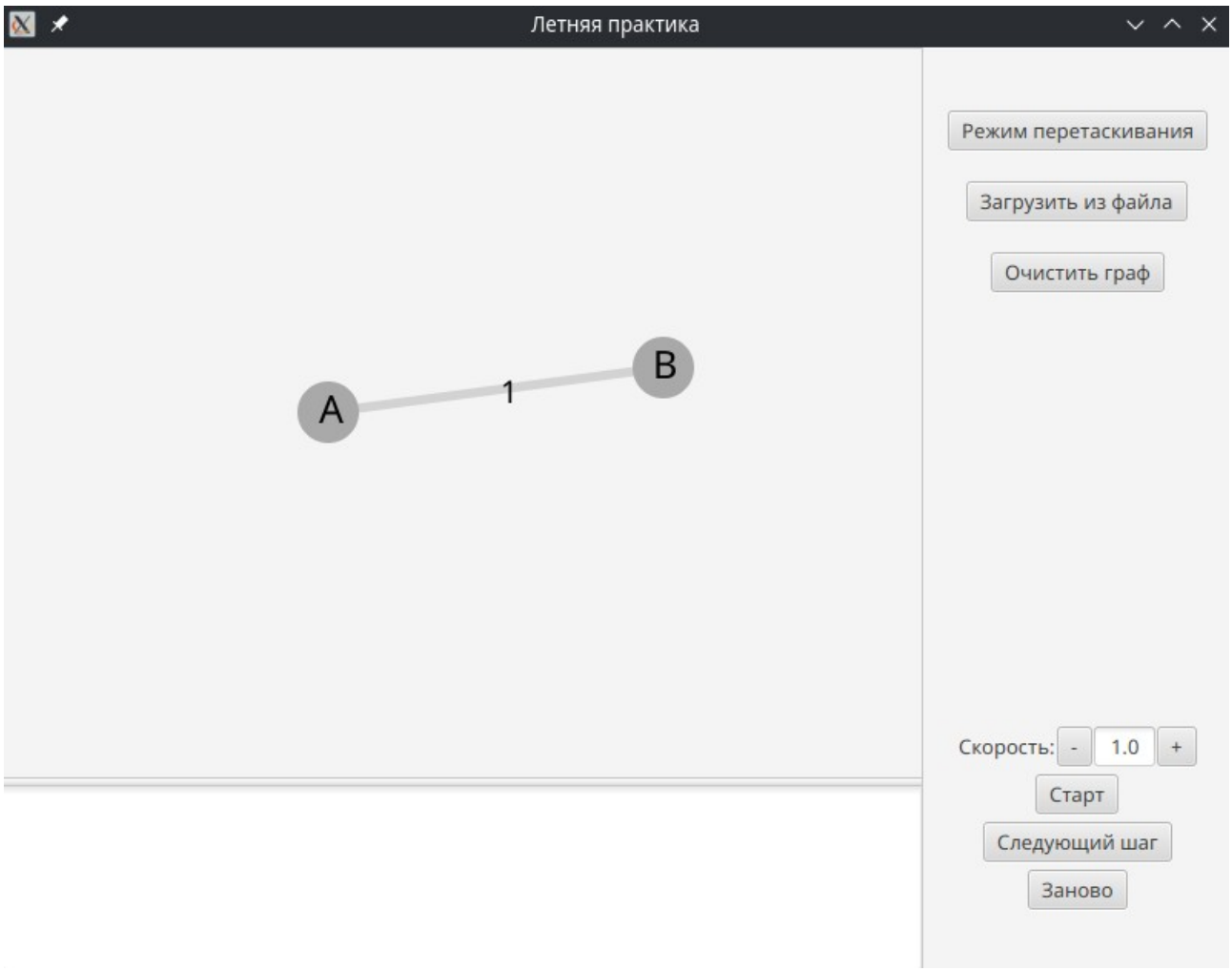


Рисунок 19 — новое ребро создано

Создание рёбер работает нормально.

3. Удаление ребра.

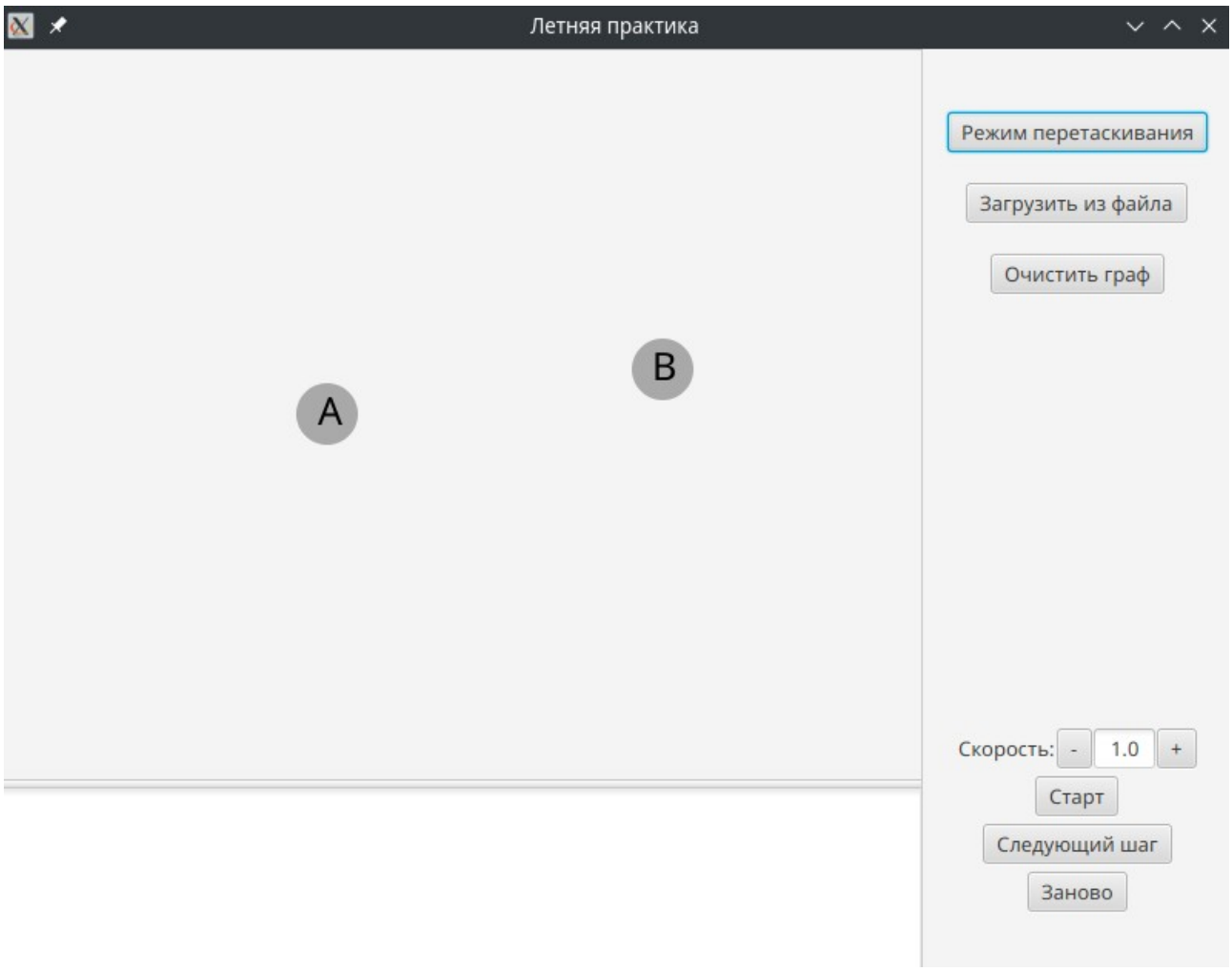


Рисунок 20 — ребро удалено

Удаление рёбер работает нормально.

4. Удаление вершины. (пусть ребро из A в B уже проведено)

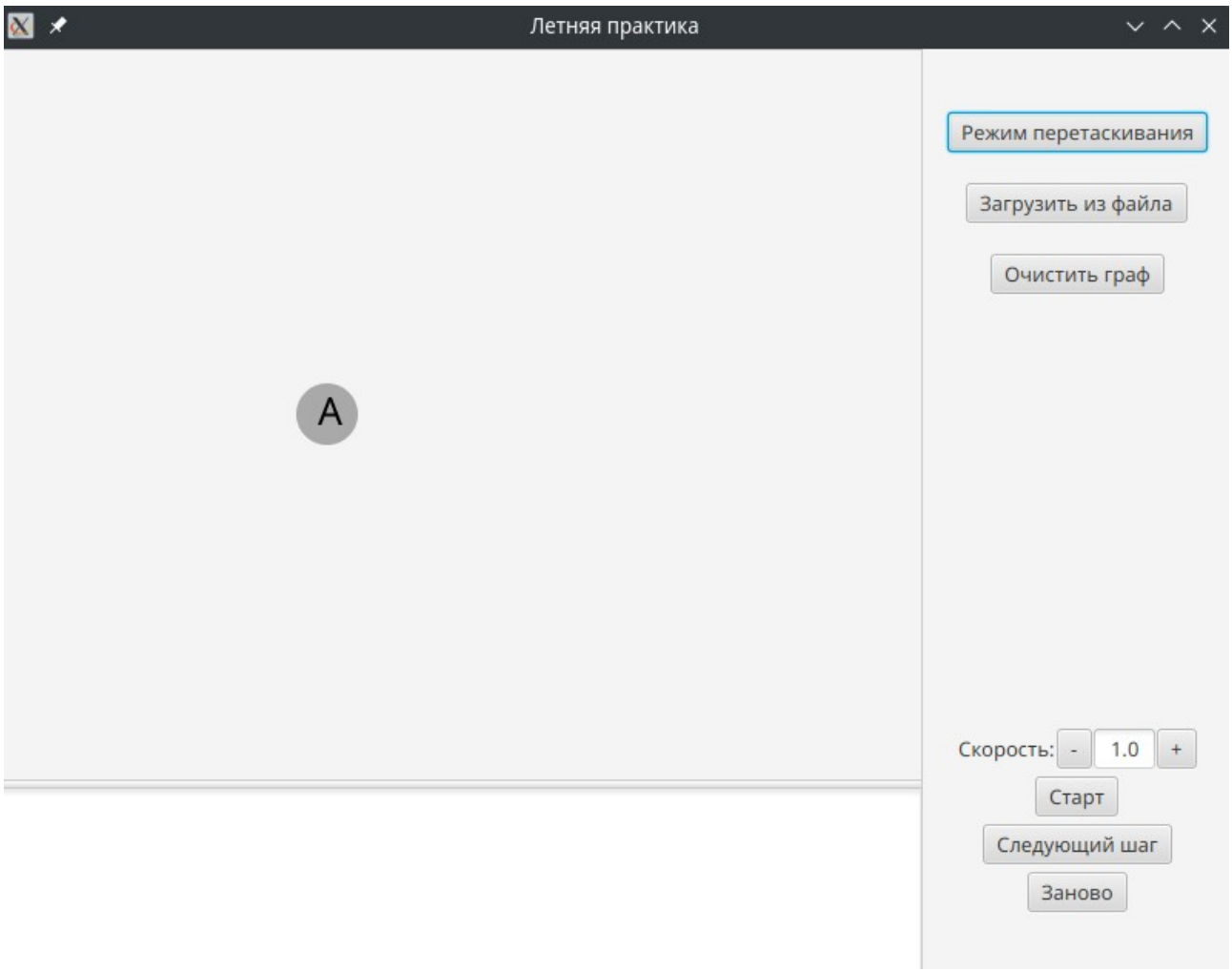


Рисунок 21 — вершина B вместе со всеми своими рёбрами удалена

Удаление вершин работает нормально.

5. Перемещение вершин.

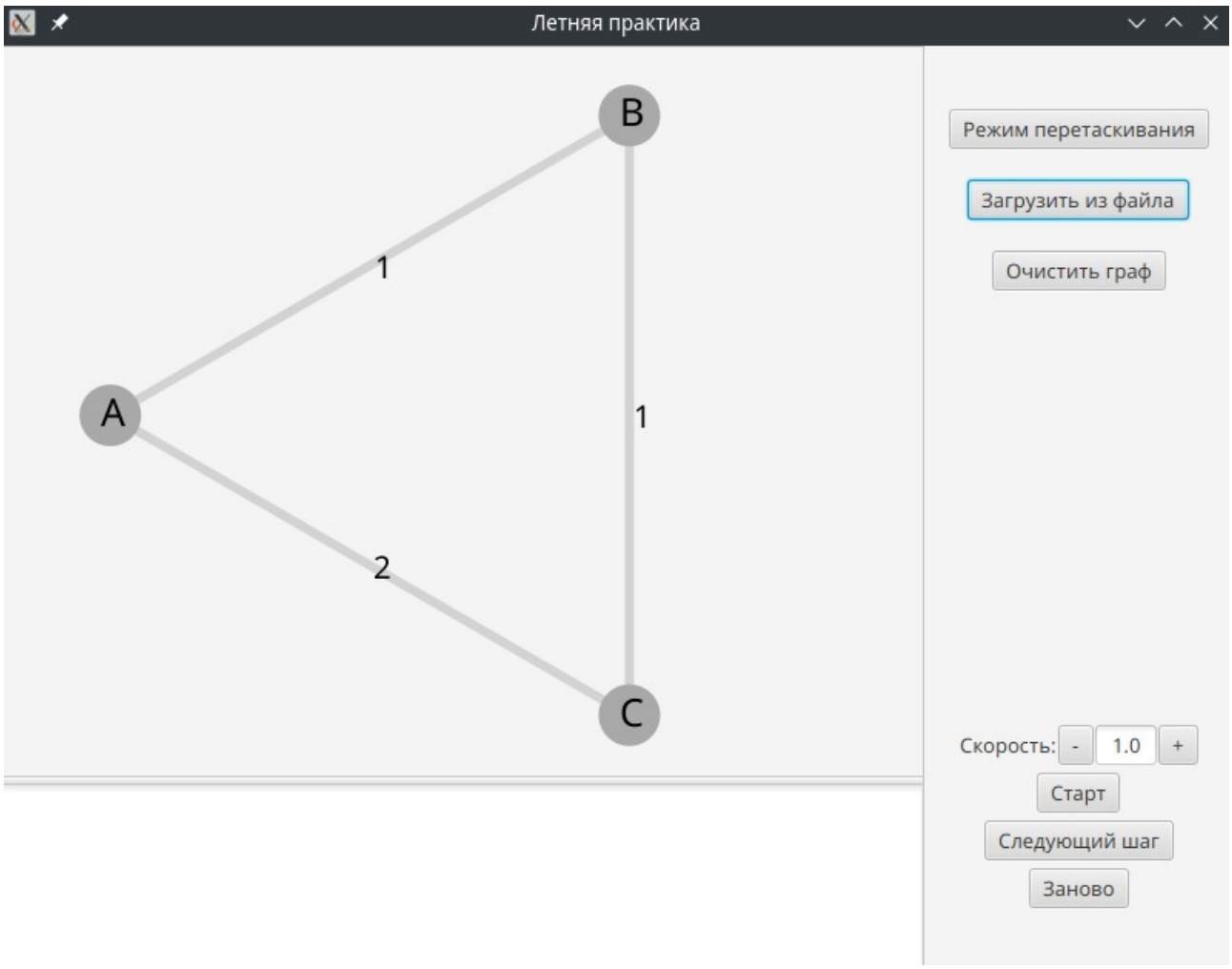


Рисунок 21 — граф до перемещения вершин

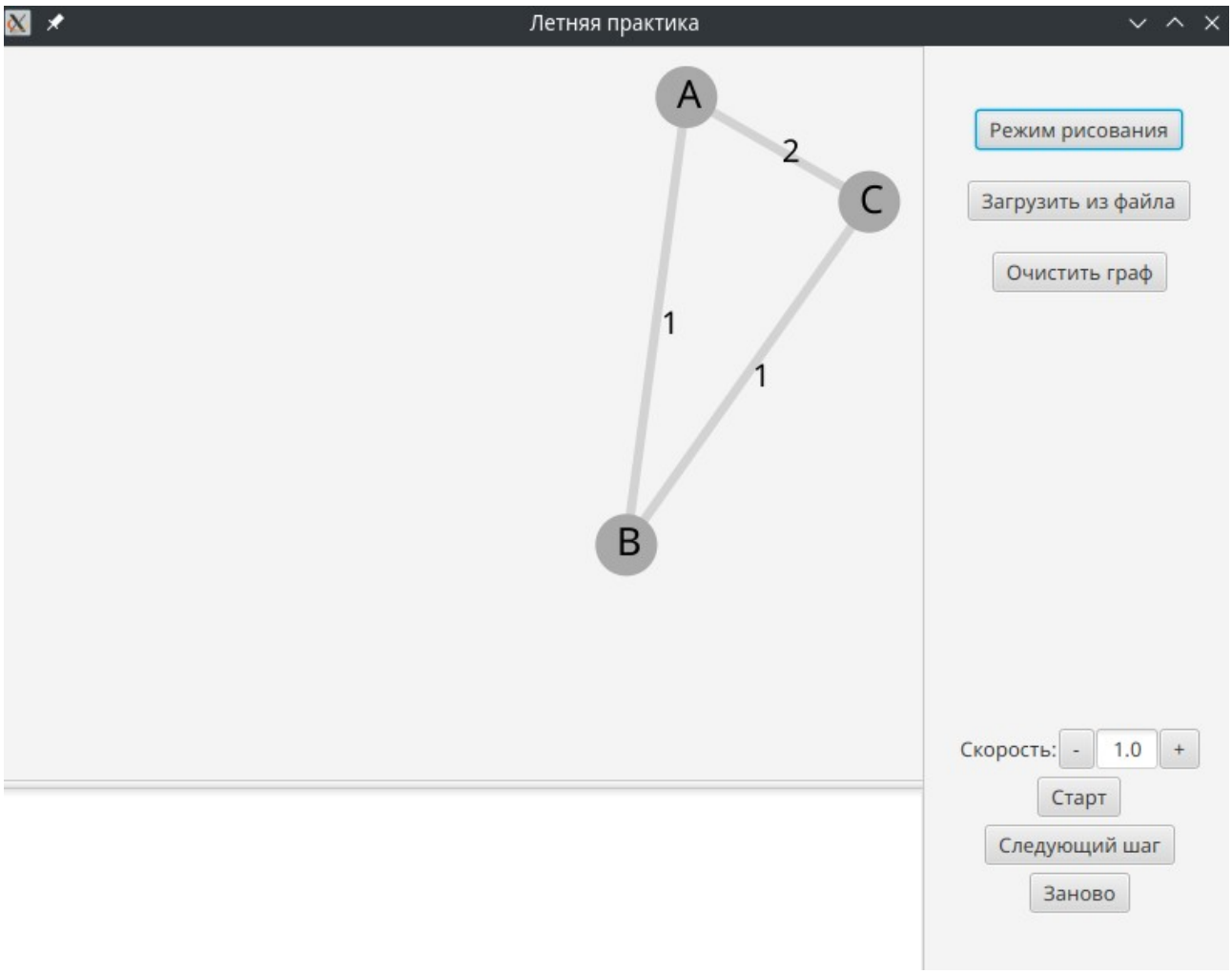


Рисунок 22 — граф после перемещения вершин

Перемещение вершин работает нормально.

7. Перемещение всего графа

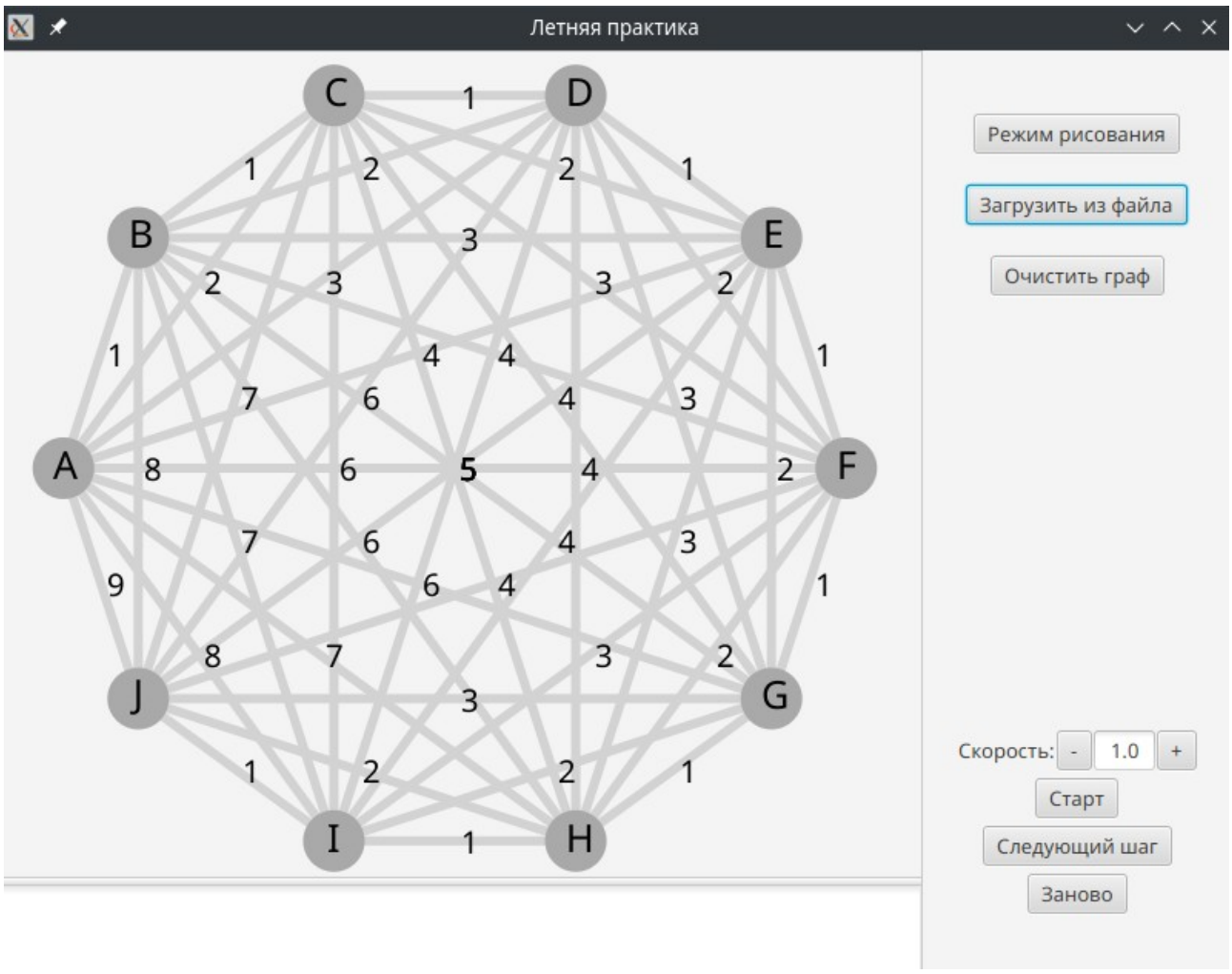


Рисунок 23 — граф до перемещения

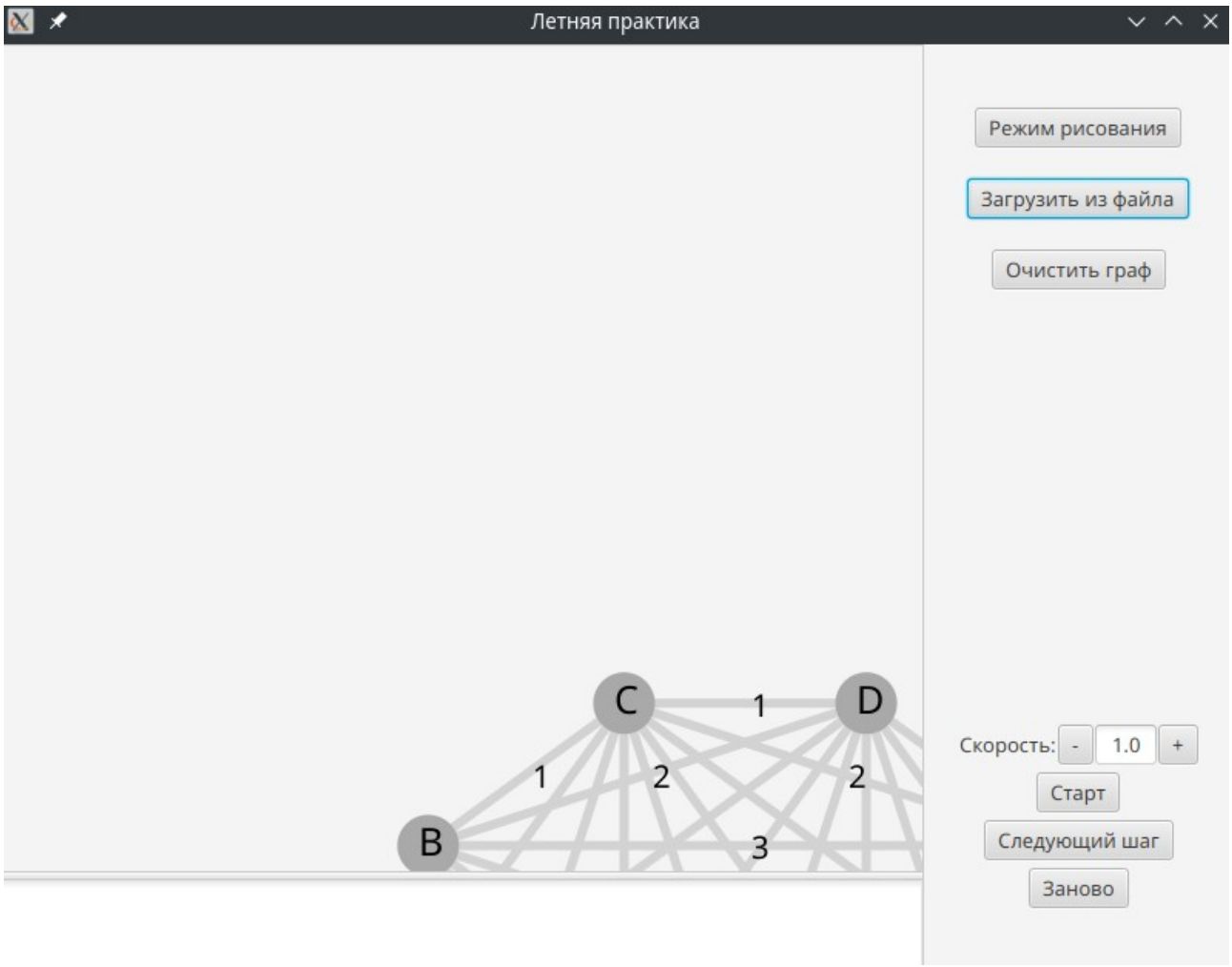


Рисунок 24 — граф после перемещения

Перемещение графа работает нормально.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы был освоен язык программирования «Java», а также разработано графическое приложение для нахождения минимального остовного дерева с помощью алгоритма Борувки. Разработка выполнялась в бригаде из трех человек, где каждый отвечал за определенную часть работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Статья об алгоритме Борувки // Wikipedia. URL: https://en.wikipedia.org/wiki/Borůvka%27s_algorithm (дата обращения: с 04.07.2022 по 11.07.2022).

2. Онлайн-курс по языку Java // Java. Базовый курс. URL: https://stepik.org/lesson/12752/step/1?unit=3102 (дата обращения: 29.07.2022).

3. Описание библиотеки JavaFX // JENKOV. URL: https://www.jenkov.com/tutorials/javafx/overview.html (дата обращения: 10.07.2022).

4. Описание библиотеки JavaFX // Введение в JavaFX. URL: https://metanit.com/java/javafx/1.1.php (дата обращения: 07.07.2022).

5. Сайт для задавания иди поиска вопросов, а также ответов на них // Stack Overflow. URL: https://stackoverflow.com (дата обращения: 04.07.2022)

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ПРИЛОЖЕНИЯ

Файл Algorithm:

```
package com.leti.summer_practice.logic;

import java.util.*;

public class Algorithm {

    private ArrayList<Graph.Node> vertices;
    private ArrayList<Graph.Edge> edges;
    private Graph temporary_graph; //второй граф для поиска МОД
    private Graph.Edge[] result; //список списков ребер,
    // которые были добавлены в компоненты связности в конце шага
    private Map<Graph.Node, Integer> hashTableNode; //хэш-таблица для
вершин и компонент
    private Map<Graph.Edge, Integer> edges_color;

    int length_components;

    Algorithm(Graph graph1) {
        vertices = graph1.get_vertices(); //все вершины графа
        edges = graph1.get_edges(); //все ребра графа
        temporary_graph = new Graph();
        length_components = vertices.size();
        edges_color = new HashMap<>();
        hashTableNode = new HashMap<>();

        for (int i = 0; i < vertices.size(); i++) { //заполенение
второго графа
            temporary_graph.create_vertex(vertices.get(i).get_name());
            hashTableNode.put(vertices.get(i), i);
        }
    }


    private static class Inner {
        boolean val;

        Inner(boolean val) {
            this.val = val;
        }
    }

    private int update_components() {
        Map<Graph.Node, Inner> all_vertices = new HashMap<>();
        ArrayList<Inner> open_vertices = new ArrayList<>();
        for (int i = 0; i < vertices.size(); i++) {
            Inner new_val = new Inner(false);
            all_vertices.put(vertices.get(i), new_val);
            open_vertices.add(new_val);
        }
        int counter = 0;
        for (int i = 0; i < open_vertices.size(); i++) {
            if (open_vertices.get(i).val) {
```

```
                continue;
            }
            dfs(all_vertices, vertices.get(i), counter);
            counter++;
        }
        return counter;
    }

    private void dfs(Map<Graph.Node, Inner> all_vertices, Graph.Node
current, int counter) {
        hashTableNode.put(current, counter);
        all_vertices.get(current).val = true;
        ArrayList<Graph.Node> neighbours =
temporary_graph.get_neighbours(current);
        for (int i = 0; i < neighbours.size(); i++) {
            if (!all_vertices.get(neighbours.get(i)).val) {
                Graph.Edge new_edge =
temporary_graph.get_edge(current.get_name(),
neighbours.get(i).get_name());
                edges_color.put(new_edge, counter);
                dfs(all_vertices, neighbours.get(i), counter);
            }
        }


    }

    public Graph.Edge[] get_new_edges() {

        if (length_components == 1) {
            return null;
        }

        result = new Graph.Edge[length_components];

        for (int i = 0; i < edges.size(); i++) {
            Graph.Edge current_edge = edges.get(i);
            int start_component =
hashTableNode.get(current_edge.get_start());
            int finish_component =
hashTableNode.get(current_edge.get_finish());

            if (start_component != finish_component) {
                if (result[start_component] == null ||
result[start_component].get_weight() > current_edge.get_weight()) {
                    result[start_component] = current_edge;
                }
                if (result[finish_component] == null ||
result[finish_component].get_weight() > current_edge.get_weight()) {
                    result[finish_component] = current_edge;
                }
            }
        }

        for (int i = 0; i < result.length; i++) {
            Graph.Edge current = result[i];
            temporary_graph.create_edge(current.get_start(),
current.get_finish(), current.get_weight());
```

```
        }

        return result;
    }

    public void next_step() {
        if (length_components == 1) {
            return;
        }
        length_components = update_components();
    }

    boolean isFinished() {
        return length_components == 1;
    }

    ArrayList<Graph.Edge> get_answer() {
        if (!isFinished()) {
            throw new RuntimeException("Algorithm is not finished yet");
        }
        return temporary_graph.get_edges();
    }

    public Integer get_vertex_color(Graph.Node vertex) {
        return hashTableNode.get(vertex);
    }

    public Integer get_edge_color(Graph.Node start_vertex, Graph.Node
finish_vertex) {
        if(!
temporary_graph.edge_exists(start_vertex.get_name(),finish_vertex.get_na
me())){
            return null;
        }
        Graph.Edge edge =
temporary_graph.get_edge(start_vertex.get_name(),
finish_vertex.get_name());

        return edges_color.get(edge);
    }
}
```

Файл Graph:

```
package com.leti.summer_practice.logic;

import java.io.*;
import java.util.*;

public class Graph {

    private ArrayList<ArrayList<Integer>> matrix;
    private Map<String, Integer> adress;
    int max_index;
    LinkedList<Integer> available_numbers;

    public Graph() {
        matrix = new ArrayList<>();
        adress = new HashMap<>();
```

```
        available_numbers = new LinkedList<>();
        max_index = 0;
    }

    public int get_vertex_count() {
        return adress.size();
    }

    public void clear() {
        matrix.clear();
        adress.clear();
        available_numbers.clear();
        max_index = 0;
    }

    public boolean is_connected() {
        Map<Graph.Node, Boolean> closed = new HashMap<>();
        Set<String> entry = adress.keySet();
        Iterator<String> iter = entry.iterator();
        if (!iter.hasNext()) {
            return true;
        }
        Node start_vertex = new Node(iter.next());
        int vertex_count = dfs(closed, start_vertex);
        return vertex_count == get_vertex_count();
    }


    private int dfs(Map<Graph.Node, Boolean> closed, Graph.Node current)
{
        closed.put(current, true);
        int res = 1;
        ArrayList<Graph.Node> neighbours = get_neighbours(current);
        for (int i = 0; i < neighbours.size(); i++) {
            if (!closed.containsKey(neighbours.get(i))) {
                res = res + dfs(closed, neighbours.get(i));
            }
        }
        return res;
    }

    private static class Graph_builder {
        Graph result;
        String[] vertex_names;

        private int correct_line_length;
        private boolean column_mode;
        private int current_line_count;

        private boolean first_iteration;

        private boolean triangle_mode;

        Graph_builder() {
            result = new Graph();
            column_mode = true;
            triangle_mode = false;
            first_iteration = true;
```

```
            current_line_count = 0;
        }

        void init_names(String[] names) {
            for (int i = 0; i < names.length; i++) {
                if (names[i].length() > 3 || !
names[i].chars().allMatch(Character::isLetter)) {
                    throw new RuntimeException("Error while reading
file:invalid vertex name");
                }
                result.create_vertex(names[i]);
            }
            vertex_names = names;
        }

        private void add_directed_edge(String start, String finish,
Integer weight) {
            int start_index = result.adress.get(start);
            int finish_index = result.adress.get(finish);
            result.matrix.get(start_index).set(finish_index, weight);
        }

        private void check_mode(String[] row) {
            if (current_line_count > result.get_vertex_count()) {
                throw new RuntimeException("Error while reading file:The
number of adjacency matrix rows does not match yhe number of columns");
            }

            if (row[0].chars().allMatch(Character::isLetter)) {
                if (!column_mode) {
                    throw new RuntimeException("Error while reading
file:Incorrect matrix");
                }

            } else {

                if (column_mode) {
                    if (first_iteration) {
                        column_mode = false;
                    } else {
                        throw new RuntimeException("Error while reading
file:Incorrect matrix");
                    }
                }
            }
            if (first_iteration) {
                if ((row.length == 1 && !column_mode) || (row.length == 2
&& column_mode)) {
                    correct_line_length = 1;
                    triangle_mode = true;
                } else {
                    correct_line_length = result.get_vertex_count();
                }
                if (column_mode) {
                    correct_line_length++;
                }
            }
            if (triangle_mode && !first_iteration) {
```

```
                correct_line_length++;
            }

            if (row.length != correct_line_length) {
                throw new RuntimeException("Error while reading
file:Incorrect row length");
            }

            if (column_mode) {
                if (!row[0].equals(vertex_names[current_line_count])) {
                    throw new RuntimeException("Error while reading file:
Incorrect  name column");
                }
            }
            first_iteration = false;
        }

        void read_row(String[] row) {
            check_mode(row);
            int start_index = column_mode ? 1 : 0;
            for (int i = start_index; i < row.length; i++) {
                if (!row[i].equals("-")) {
                    try {
                        int new_val = Integer.parseInt(row[i]);
                        String start = vertex_names[current_line_count];
                        String finish = vertex_names[i - start_index];
                        if (!triangle_mode) {
                            add_directed_edge(start, finish, new_val);
                        } else {
                            result.create_edge(result.get_vertex(start),
result.get_vertex(finish), new_val);
                        }
                    } catch (NumberFormatException ex) {
                        throw new RuntimeException("Error while reading
file:Invalid edge weight");
                    }
                }
            }
            current_line_count++;
        }

        Graph get_result() {
            if (column_mode) {
                current_line_count++;
            }
            if (current_line_count != correct_line_length) {
                throw new RuntimeException("Error while reading file:
Adjacency matrix is not square");
            }
            if (!result.symmetry_check()) {
                throw new RuntimeException("Error while reading file:The
Adjacency matrix is not symmetrical");
            }
            return result;
        }


    }
```

```
    boolean edge_exists(String start, String finish) {
        if (!adress.containsKey(start) || !adress.containsKey(finish)) {
            return false;
        }
        int start_index = adress.get(start);
        int finish_index = adress.get(finish);
        return matrix.get(start_index).get(finish_index) != null;
    }

    boolean vertex_exists(String name) {
        return adress.containsKey(name);
    }

    private static String[] check_first(String[] values) {
        if (values[0].equals("")) {
            ArrayList<String> temp = new
ArrayList<>(Arrays.asList(values));
            temp.remove(0);
            return temp.toArray(new String[temp.size()]);
        }
        return values;
    }

    static public Graph read_file(File file) {
        try (FileReader reader = new FileReader(file); BufferedReader
buffer = new BufferedReader(reader)) {
            String line = buffer.readLine();
            if (line == null) {
                return new Graph();
            }
            Graph_builder builder = new Graph_builder();
            String[] names = line.split(" +");
            names = check_first(names);
            builder.init_names(names);
            line = buffer.readLine();
            while (line != null) {
                String[] new_values = line.split(" +");
                new_values = check_first(new_values);
                builder.read_row(new_values);
                line = buffer.readLine();
            }
            return builder.get_result();
        } catch (IOException e) {
            throw new RuntimeException("error while opening file");
        }
    }

    private boolean symmetry_check() {
        for (int i = 0; i < matrix.size(); i++) {
            for (int j = 0; j < matrix.size(); j++) {
                if (!Objects.equals(matrix.get(i).get(j),
matrix.get(j).get(i))) {
                    return false;
                }
                if (i == j && matrix.get(i).get(j) != null) {
                    return false;
                }
```

```
            }
        }
        return true;
    }


    public Node create_vertex(String name) {
        if (adress.containsKey(name)) {
            throw new UnsupportedOperationException("Vertex with same
name already exists");
        }
        Node new_node = new Node(name);
        int new_index;
        if (available_numbers.size() > 0) {
            new_index = available_numbers.removeLast();
            adress.put(name, new_index);
        } else {
            new_index = max_index;
            max_index++;
            adress.put(name, new_index);
            resize_matrix();
        }

        return new_node;
    }

    public void delete_vertex(Node vertex) {
        if (!adress.containsKey(vertex.name)) {
            throw new UnsupportedOperationException("Invalid vertex
name");
        }
        int vertex_index = adress.get(vertex.name);
        adress.remove(vertex.name);
        available_numbers.addLast(vertex_index);
        for (int i = 0; i < matrix.size(); i++) {
            matrix.get(i).set(vertex_index, null);
            matrix.get(vertex_index).set(i, null);
        }
    }

    public ArrayList<Node> get_vertices() {
        ArrayList<Node> res = new ArrayList<>();
        for (String name : adress.keySet()) {
            Node current = new Node(name);
            res.add(current);
        }
        return res;
    }

    public Node get_vertex(String name) {
        if (!adress.containsKey(name)) {
            throw new UnsupportedOperationException("there is no such
vertex of the graph");
        }
        return new Node(name);
    }

    public Edge get_edge(String start, String finish) {
```

```
        if (!edge_exists(start, finish)) {
            throw new UnsupportedOperationException("Edge does not
exist");

        }
        int start_index = adress.get(start);
        int finish_index = adress.get(finish);
        int weight = matrix.get(start_index).get(finish_index);
        return new Edge(new Node(start), new Node(finish), weight);
    }

    public Edge create_edge(Node start, Node finish, int weight) {
        if (!adress.containsKey(start.name) || !
adress.containsKey(finish.name)) {
            throw new UnsupportedOperationException("Invalid vertices for
edge");
        }
        int start_index = adress.get(start.name);
        int finish_index = adress.get(finish.name);
        if (matrix.get(start_index).get(finish_index) != null) {
            if (matrix.get(start_index).get(finish_index) == weight) {
                return new Edge(start, finish, weight);
            }
            throw new UnsupportedOperationException("such an edge already
exists");
        }
        matrix.get(start_index).set(finish_index, weight);
        matrix.get(finish_index).set(start_index, weight);
        return new Edge(start, finish, weight);
    }

    public void delete_edge(Edge to_delete) {
        if (!adress.containsKey(to_delete.start.name) || !
adress.containsKey(to_delete.finish.name)) {
            throw new UnsupportedOperationException("Edge does not
exist");
        }
        int start_index = adress.get(to_delete.start.name);
        int finish_index = adress.get(to_delete.finish.name);
        matrix.get(start_index).set(finish_index, null);
        matrix.get(finish_index).set(start_index, null);
    }

    public ArrayList<Edge> get_edges() {
        ArrayList<Edge> res = new ArrayList<>();
        for (Map.Entry<String, Integer> entry : adress.entrySet()) {
            int current_vertex = entry.getValue();
            for (Map.Entry<String, Integer> entry1 : adress.entrySet()) {
                int current_index = entry1.getValue();
                if (current_vertex < current_index ||
matrix.get(current_vertex).get(current_index) == null) {
                    continue;
                }
                Node start = new Node(entry.getKey());
                Node finish = new Node(entry1.getKey());
                int weight =
matrix.get(current_vertex).get(current_index);
                res.add(new Edge(start, finish, weight));
```

```
            }
        }
        return res;
    }

    ArrayList<Node> get_neighbours(Node current) {
        if (!adress.containsKey(current.name)) {
            throw new UnsupportedOperationException("there is no such
vertex of the graph");
        }
        ArrayList<Node> res = new ArrayList<>();
        int current_index = adress.get(current.name);
        for (Map.Entry<String, Integer> entry : adress.entrySet()) {
            if (entry.getValue() == current_index ||
matrix.get(current_index).get(entry.getValue()) == null) {
                continue;
            }
            Node new_node = new Node(entry.getKey());
            res.add(new_node);
        }
        return res;
    }


    private void resize_matrix() {
        for (int i = 0; i < matrix.size(); i++) {
            matrix.get(i).add(null);
        }
        ArrayList<Integer> new_list = new ArrayList<>();
        for (int i = 0; i <= matrix.size(); i++) {
            new_list.add(null);
        }
        matrix.add(new_list);
    }


    public static class Node {
        private String name;

        private Node(String name) {
            this.name = name;
        }

        public String get_name() {
            return name;
        }

        @Override //PґP»CЏ C,PµCЃC,PsPI - PїPsC,PsPj CґPґP°P»PёPj
        public String toString() {
            return name;
        }

        @Override
        public boolean equals(Object other) {
            if (other == this) {
                return true;
            }
```

```
            if (other instanceof Node) {
                Node current = (Node) other;
                if (current.name.equals(this.name)) {
                    return true;
                }
            }

            return false;
        }

        @Override
        public int hashCode() {
            return name.hashCode();
        }
    }

    public static class Edge {

        private Node start;
        private Node finish;
        private int weight;

        private Edge(Node start, Node finish, int weight) {
            this.start = start;
            this.finish = finish;
            this.weight = weight;
        }

        public Node get_start() {
            return start;
        }

        public Node get_finish() {
            return finish;
        }

        public int get_weight() {
            return weight;
        }

        public boolean equals(Object other) {
            if (this == other) {
                return true;
            }
            if (other instanceof Edge) {
                Edge other_edge = (Edge) other;
                if ((start.equals(other_edge.start) &&
finish.equals(other_edge.finish)) || (finish.equals(other_edge.start) &&
start.equals(other_edge.finish))) {
                    if (weight == other_edge.weight) {
                        return true;
                    }
                }
            }
            return false;
        }

        @Override
```

```
        public int hashCode() {
            return start.hashCode() + finish.hashCode();
        }

        @Override //РґР»СЏ С,РµСЃС,PsPI = РїPsC,PsPj СѓРґР°Р»РёPj
        public String toString() {
            return new String("( " + start.name + " " + finish.name + " "
+ weight + " )");
        }
    }
}
```

Файл Logic:

```
package com.leti.summer_practice.logic;

import java.io.File;
import java.util.ArrayList;

public class Logic implements LogicInterface {

    private Graph graph;
    private Algorithm algorithm;

    private int current_edge;
    Graph.Edge[] current_step_edges;

    public Logic() {
        graph = new Graph();
        current_edge = 0;
        current_step_edges = null;
    }

    @Override
    public void removeVertex(String name) {
        Graph.Node delete = graph.get_vertex(name);
        graph.delete_vertex(delete);
    }

    @Override
    public void addVertex(String name) {
        graph.create_vertex(name);
    }

    @Override
    public void addEdge(String start, String finish, int weight) {
        Graph.Node start_vertex = graph.get_vertex(start);
        Graph.Node finish_vertex = graph.get_vertex(finish);
        graph.create_edge(start_vertex, finish_vertex, weight);
    }

    @Override
    public void removeEdge(String start, String finish) {
        graph.delete_edge(graph.get_edge(start, finish));
    }

    @Override
    public VertexInfo getVertexInfo(String name) {
```

```
        Integer color;
        if (algorithm == null) {
            color = null;
        } else {
            Graph.Node vertex = graph.get_vertex(name);
            color = algorithm.get_vertex_color(vertex);
        }

        return new VertexInfo(name, color);
    }

    @Override
    public EdgeInfo getEdgeInfo(String start, String finish) {

        Integer color;
        if (algorithm == null) {
            color = null;
        } else {
            Graph.Node start_vertex = graph.get_vertex(start);
            Graph.Node finish_vertex = graph.get_vertex(finish);
            color = algorithm.get_edge_color(start_vertex,
finish_vertex);
        }
        int weight = graph.get_edge(start, finish).get_weight();
        return new EdgeInfo(start, finish, weight, color);
    }

    @Override
    public ArrayList<VertexInfo> getVertices() {
        ArrayList<Graph.Node> nodes = graph.get_vertices();
        ArrayList<VertexInfo> vertexInfos = new
ArrayList<>(nodes.size());
        for (Graph.Node node : nodes) {
            VertexInfo vertexInfo = new VertexInfo();
            vertexInfo.name = node.get_name();
            if (algorithm == null) {
                vertexInfo.color = null;
            } else {
                vertexInfo.color = algorithm.get_vertex_color(node);
            }
            vertexInfos.add(vertexInfo);
        }
        return vertexInfos;
    }

    @Override
    public ArrayList<EdgeInfo> getEdges() {
        ArrayList<Graph.Edge> edges = graph.get_edges();
        ArrayList<EdgeInfo> edge_infos = new ArrayList<>(edges.size());
        for (Graph.Edge edge : edges) {
            EdgeInfo edge_info = new EdgeInfo();
            edge_info.start = edge.get_start().get_name();
            edge_info.finish = edge.get_finish().get_name();
            edge_info.weight = edge.get_weight();
            if (algorithm == null) {
                edge_info.color = null;
            } else {
```

```
                edge_info.color =
algorithm.get_edge_color(edge.get_start(), edge.get_finish());
            }
            edge_infos.add(edge_info);
        }
        return edge_infos;
    }

    @Override
    public void loadFile(File file) {
        killAlgorithm();
        graph = Graph.read_file(file);
    }

    @Override
    public void startAlgorithm() {
        if (algorithm != null) {
            return;
        }
        if (!graph.is_connected()) {
            throw new RuntimeException("Graph is not connected");
        }
        algorithm = new Algorithm(graph);
        current_step_edges = algorithm.get_new_edges();
        current_edge = 0;
    }

    @Override
    public EdgeInfo getNewEdge() {
        if (algorithm == null) {
            throw new RuntimeException("Algorithm is not started");
        }
        if (current_step_edges == null || current_edge ==
current_step_edges.length) {
            return null;
        }
        EdgeInfo new_edge = new EdgeInfo();
        new_edge.start =
current_step_edges[current_edge].get_start().get_name();
        new_edge.finish =
current_step_edges[current_edge].get_finish().get_name();
        new_edge.color = current_edge;
        new_edge.weight = current_step_edges[current_edge].get_weight();
        current_edge++;
        return new_edge;
    }

    @Override
    public void nextBigStep() {
        if (algorithm == null) {
            throw new RuntimeException("Algorithm is not started");
        }
        algorithm.next_step();
        current_step_edges = algorithm.get_new_edges();
        current_edge = 0;
    }

    @Override
```

```
    public boolean isAlgorithmFinished() {
        return algorithm != null && algorithm.isFinished();
    }

    @Override
    public void killAlgorithm() {
        algorithm = null;
    }

    @Override
    public boolean isAlgorithmStarted() {
        return algorithm != null;
    }


    @Override
    public ArrayList<EdgeInfo> getAnswer() {
        if (algorithm == null) {
            throw new RuntimeException("Algorithm is not finished yet");
        }
        ArrayList<Graph.Edge> answer = algorithm.get_answer();
        ArrayList<EdgeInfo> res = new ArrayList<>();
        for (Graph.Edge edge : answer) {
            EdgeInfo new_edge = new EdgeInfo();
            new_edge.start = edge.get_start().get_name();
            new_edge.finish = edge.get_finish().get_name();
            if (algorithm == null) {
                new_edge.color = null;
            } else {
                new_edge.color =
algorithm.get_edge_color(edge.get_start(), edge.get_finish());
            }
            new_edge.weight = edge.get_weight();
            res.add(new_edge);
        }
        return res;
    }

    @Override
    public boolean isGraphEmpty() {
        return graph.get_vertex_count() == 0;
    }

    @Override
    public void clearGraph() {
        killAlgorithm();
        graph.clear();
    }

    @Override
     public boolean edgeExists(String start, String finish) {
        return graph.edge_exists(start, finish);
    }

    @Override
    public boolean vertexExists(String name) {
        return graph.vertex_exists(name);
    }
```

```
}
```

Файл LogicInterface:

```
package com.leti.summer_practice.logic;

import java.io.File;
import java.util.ArrayList;

public interface LogicInterface {

    class VertexInfo {

        public String name;
        public Integer color;

        public VertexInfo() {
        }

        public VertexInfo(String name, Integer color) {
            this.name = name;
            this.color = color;
        }
    }

    class EdgeInfo {

        public String start;
        public String finish;
        public int weight;
        public Integer color;

        public EdgeInfo() {
        }

        public EdgeInfo(String start, String finish, int weight, Integer
color) {
            this.start = start;
            this.finish = finish;
            this.weight = weight;
            this.color = color;
        }

        public String toStringWithoutWeight() {
            if (start.compareTo(finish) > 0)
                return finish + " --- " + start;
            return start + " --- " + finish;
        }
    }

    void removeVertex(String name);

    void addVertex(String name);

    void addEdge(String start, String finish, int weight);

    void removeEdge(String start, String finish);
```

```
    VertexInfo getVertexInfo(String name);

    EdgeInfo getEdgeInfo(String start, String finish);

    ArrayList<VertexInfo> getVertices();

    ArrayList<EdgeInfo> getEdges();

    void loadFile(File file);

    void startAlgorithm();

    EdgeInfo getNewEdge();

    void nextBigStep();

    boolean isAlgorithmFinished();

    ArrayList<EdgeInfo> getAnswer();

    boolean isAlgorithmStarted();

    boolean edgeExists(String start, String finish);

    boolean vertexExists(String name);

    void killAlgorithm();

    boolean isGraphEmpty();

    void clearGraph();
}
```

Файл GraphCanvas:

```
package com.leti.summer_practice.gui.prog;

import com.leti.summer_practice.R;
import com.leti.summer_practice.gui.lib.VectorCanvas;
import com.leti.summer_practice.logic.LogicInterface;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.event.Event;
import javafx.event.EventHandler;
import javafx.scene.control.Alert;
import javafx.scene.control.TextFormatter;
import javafx.scene.control.TextInputDialog;
import javafx.scene.input.MouseButton;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.scene.text.TextBoundsType;
import javafx.util.Pair;
import javafx.util.StringConverter;

import java.util.*;
import java.util.regex.Pattern;
```

```
import java.util.stream.Collectors;


public class GraphCanvas extends VectorCanvas {


    public static final double DEFAULT_CANVAS_WIDTH = 200; // relative
    public static final double DEFAULT_CANVAS_HEIGHT = 200; // relative
    public static final double DEFAULT_CANVAS_CAMERA_X = -100; //
relative
    public static final double DEFAULT_CANVAS_CAMERA_Y = -100; //
relative

    public static final double DEFAULT_VERTEX_RADIUS = 20; // pixels
    public static final double DEFAULT_VERTEX_NAME_TEXT_SIZE = 15; //
pixels

    public static final double DEFAULT_EDGE_STROKE_WIDTH = 6; // pixels
    public static final double DEFAULT_EDGE_WEIGHT_TEXT_SIZE = 12; //
pixels


    private static final Color DEFAULT_VERTEX_COLOR = Color.DARKGRAY;
    private static final Color DEFAULT_EDGE_COLOR = Color.LIGHTGRAY;


    public enum GraphMode {
        DRAWING,
        MOVING;

        public static final GraphMode DEFAULT_GRAPH_MODE = MOVING;
    }


    private class CircleEvents {

        public static final Color SELECTED_VERTEX_COLOR = Color.RED;

        private double mouseX, mouseY;
        private Circle selectedCircle = null;
        private String selectedCircleName = null;

        public final EventHandler<MouseEvent> onMousePressedEventHandler
= event -> {

            if (event.getButton() != MouseButton.PRIMARY)
                return;

            mouseX = event.getX();
            mouseY = event.getY();
        };

        public final EventHandler<MouseEvent> onMouseDraggedEventHandler
= event -> {

            if (event.getButton() != MouseButton.PRIMARY)
                return;
            if (graphMode != GraphMode.MOVING)
```

```
                return;

            Circle circle = (Circle) event.getSource();
            circle.setCenterX(circle.getCenterX() + (event.getX() -
mouseX));
            circle.setCenterY(circle.getCenterY() + (event.getY() -
mouseY));
            mouseX = event.getX();
            mouseY = event.getY();
            event.consume();
        };

        public final EventHandler<MouseEvent> onMouseClickedEventHandler
= event -> {

            if (graphMode != GraphMode.DRAWING) {
                event.consume();
                return;
            }

            Circle circle = (Circle) event.getSource();
            String name = reversedVerticesMap.get(circle);

            if (event.getButton() == MouseButton.PRIMARY) {

                if (selectedCircle == null) {
                    selectedCircle = circle;
                    selectedCircleName = name;
                    redraw();
                } else {

                    if (logic.edgeExists(selectedCircleName, name)) {
                        Alert alert = new Alert(Alert.AlertType.ERROR);
                        alert.setTitle(null);

alert.setHeaderText(R.string("edge_between_this_vertices_already_exists")
);
                        alert.setContentText(null);
                        alert.showAndWait();
                        event.consume();
                        return;
                    }

                    TextInputDialog textInputDialog = new
TextInputDialog();
                    textInputDialog.setTitle(null);

textInputDialog.setHeaderText(R.string("weight_input_dialog_header"));

//
DoubleNumberTextField.makeTextFieldAcceptOnlyDoubleNumbers(textInputDialo
g.getEditor());

                    final Pattern validEditingState =
Pattern.compile("(([1-9]\\d*)|0)?");

                    textInputDialog.getEditor().setTextFormatter(
```

```
                            new TextFormatter<>(
                                    new StringConverter<>() {
                                        @Override
                                        public String toString(Integer i)
{
                                            return Integer.toString(i);
                                        }
                                        @Override
                                        public Integer fromString(String
s) {
                                            if (s.isEmpty())
                                                return 0;
                                            return Integer.valueOf(s);
                                        }
                                    },
                                    1,
                                    change -> {
                                        String newText =
change.getControlNewText();
                                        if
(validEditingState.matcher(newText).matches())
                                            return change;
                                        else
                                            return null;
                                    }
                            )
                    );

                    Optional<String> optionalWeight =
textInputDialog.showAndWait();
                    if (optionalWeight.isEmpty()) {
                        event.consume();
                        return;
                    }

                    Integer weight =
Integer.valueOf(optionalWeight.get());

                    logic.addEdge(selectedCircleName, name, weight);

                    addEdge(selectedCircleName, name, weight);

                    unselectVertex();

                    redraw();
                }

            } else if (event.getButton() == MouseButton.SECONDARY) {

                if (name.equals(selectedCircleName)) {
                    unselectVertex();
                }

                logic.removeVertex(name);

                verticesMap.remove(name);
                reversedVerticesMap.remove(circle);
```

```
                for (Iterator<Pair<String,String>> iterator =
edgesMap.keySet().iterator(); iterator.hasNext();) {
                    Pair<String,String> key = iterator.next();
                    if (key.getKey().equals(name) ||
key.getValue().equals(name)) {
                        Line line = edgesMap.get(key);
                        reversedEdgesMap.remove(line);
                        erase(line);
                        iterator.remove();
                        Text text = edgesTextsMap.get(key);
                        edgesTextsMap.remove(key);
                        erase(text);
                    }
                }

                erase(circle);

                Text text = verticesTextsMap.get(name);
                verticesTextsMap.remove(name);

                erase(text);
            }

            event.consume();
        };

        public Circle getSelectedCircle() {
            return selectedCircle;
        }

        public String getSelectedCircleName() {
            return selectedCircleName;
        }

        public boolean isVertexSelected() {
            return selectedCircle != null;
        }

        public void unselectVertex() {
            selectedCircle = null;
            selectedCircleName = null;
        }
    }


    private class LineEvents {

        public final EventHandler<MouseEvent> onMouseClickedEventHandler
= event -> {

            if (event.getButton() != MouseButton.SECONDARY)
                return;

            Line line = (Line) event.getSource();
            Pair<String,String> startFinish = reversedEdgesMap.get(line);

            logic.removeEdge(startFinish.getKey(),
startFinish.getValue());
```

```
            edgesMap.remove(startFinish);
            reversedEdgesMap.remove(line);

            erase(line);

            Text text = edgesTextsMap.get(startFinish);
            edgesTextsMap.remove(startFinish);

            erase(text);

            event.consume();
        };
    }


    private static Color getColorByInt(int n) {
        class Colors {
            private static final ArrayList<Color> colors
                    = new ArrayList<>(
                    List.of(
                            Color.GREEN, Color.ORANGE, Color.YELLOW,
                            Color.BLUE, Color.CYAN, Color.BROWN,
                            Color.AQUAMARINE, Color.SALMON, Color.PURPLE,
                            Color.PINK, Color.YELLOWGREEN, Color.OLIVE
                    )
            );
            private Color getColorByInt(int n) {
                while (n >= colors.size())
                    colors.add(Color.color(Math.random(), Math.random(),
Math.random()));
                return colors.get(n);
            }
        }
        return new Colors().getColorByInt(n);
    }

    public static Color getVertexColorByInt(Integer n) {
        if (n == null)
            return DEFAULT_VERTEX_COLOR;
        return getColorByInt(n);
    }

    public static Color getEdgeColorByInt(Integer n) {
        if (n == null)
            return DEFAULT_EDGE_COLOR;
        return getColorByInt(n);
    }


    private LogicInterface logic;


    private final Map<String,Circle> verticesMap = new HashMap<>();
    private final Map<Circle,String> reversedVerticesMap = new
HashMap<>();

    private final Map<String, Text> verticesTextsMap = new HashMap<>();
```

```
    private final Map<Pair<String,String>,Line> edgesMap = new
HashMap<>();
    private final Map<Line,Pair<String,String>> reversedEdgesMap = new
HashMap<>();

    private final Map<Pair<String,String>,Text> edgesTextsMap = new
HashMap<>();


    private final Set<LogicInterface.EdgeInfo> specialColorEdges = new
TreeSet<>(
            Comparator.comparing((LogicInterface.EdgeInfo o) -> o.start)
                    .thenComparing(o -> o.finish)
                    .thenComparingInt(o -> o.weight));
    private static final Color SPECIAL_EDGE_COLOR = Color.RED;

    private final CircleEvents circleEvents = new CircleEvents();
    private final LineEvents lineEvents = new LineEvents();

    private GraphMode graphMode = GraphMode.DEFAULT_GRAPH_MODE;


    public GraphCanvas() {

        addEventHandler(MouseEvent.ANY, new EventHandler<>() {

            private boolean dragging = false;

            @Override
            public void handle(MouseEvent event) {
                if (event.getEventType() == MouseEvent.MOUSE_PRESSED) {
                    dragging = false;
                } else if (event.getEventType() ==
MouseEvent.DRAG_DETECTED) {
                    dragging = true;
                } else if (event.getEventType() ==
MouseEvent.MOUSE_CLICKED) {
                    if (dragging)
                        return;
                    if (circleEvents.isVertexSelected()) {
                        circleEvents.unselectVertex();
                        redraw();
                    } else if (graphMode == GraphMode.DRAWING) {
                        if (event.getButton() != MouseButton.PRIMARY)
                            return;
                        TextInputDialog textInputDialog = new
TextInputDialog();
                        textInputDialog.setTitle(null);

textInputDialog.setHeaderText(R.string("name_of_vertex_input_dialog_heade
r"));
                        Optional<String> optionalName =
textInputDialog.showAndWait();
                        if (optionalName.isEmpty())
                            return;
                        String name = optionalName.get();
```

```
                        if (name.isEmpty()) {
                            Alert alert = new
Alert(Alert.AlertType.INFORMATION);
                            alert.setTitle(null);

alert.setHeaderText(R.string("name_must_not_be_empty_alert_header"));
                            alert.setContentText(null);
                            alert.showAndWait();
                            return;
                        }
                        if (verticesMap.containsKey(name)) {
                            Alert alert = new
Alert(Alert.AlertType.INFORMATION);
                            alert.setTitle(null);

alert.setHeaderText(R.string("name_already_exists_alert_header"));
                            alert.setContentText(null);
                            alert.showAndWait();
                            return;
                        }
                        logic.addVertex(name);
                        addVertex(
                                name,
                                (event.getX() -
getContent().getTranslateX()) / getScale(),
                                (event.getY() -
getContent().getTranslateY()) / getScale()
                        );
                        redraw();
                    }
                }
            }
        });

        setDrawer(canvas -> {
            // draw all gray
            drawAll(edgesMap.values()
                    .stream()
                    .filter(
                            line ->
line.getStroke().equals(DEFAULT_EDGE_COLOR)
                    ).collect(Collectors.toList()));
            // draw all colored
            drawAll(edgesMap.values()
                    .stream()
                    .filter(
                            line -> !
line.getStroke().equals(DEFAULT_EDGE_COLOR)
                    ).collect(Collectors.toList()));
            drawAll(edgesTextsMap.values());
            drawAll(verticesMap.values());
            drawAll(verticesTextsMap.values());
        });
    }


    @Override
```

```
    public void redraw() {
        notifyColorsChanged();
        super.redraw();
    }

    public LogicInterface getLogic() {
        return logic;
    }
    public void setLogic(LogicInterface logic) {
        this.logic = logic;
        initializeWithLogic(logic);
    }

    private void initializeWithLogic(LogicInterface logic) {

        verticesMap.clear();
        reversedVerticesMap.clear();
        verticesTextsMap.clear();
        edgesMap.clear();
        reversedEdgesMap.clear();
        edgesTextsMap.clear();

        {
            ArrayList<LogicInterface.VertexInfo> vertices =
logic.getVertices();
            double angle = Math.PI;
            double step = 2 * Math.PI / vertices.size();
            for (LogicInterface.VertexInfo vertex : vertices) {
                addVertex(
                        vertex.name,
                        0.9 * DEFAULT_CANVAS_WIDTH / 2 * Math.cos(angle),
                        0.9 * DEFAULT_CANVAS_HEIGHT / 2 * Math.sin(angle)
                );
                angle += step;
            }
        }

        for (LogicInterface.EdgeInfo edge : logic.getEdges())
            addEdge(edge.start, edge.finish, edge.weight);

        notifyColorsChanged();
    }

    private void addVertex(String name, double x, double y) {

        Circle circle = new Circle();
        circle.setCenterX(x);
        circle.setCenterY(y);
        circle.setRadius(DEFAULT_VERTEX_RADIUS);
        circle.radiusProperty().bind(new
SimpleDoubleProperty(DEFAULT_VERTEX_RADIUS).divide(scaleProperty()));


circle.setOnMousePressed(circleEvents.onMousePressedEventHandler);

circle.setOnMouseDragged(circleEvents.onMouseDraggedEventHandler);
```

```
circle.setOnMouseClicked(circleEvents.onMouseClickedEventHandler);

        verticesMap.put(name, circle);
        reversedVerticesMap.put(circle, name);

        Text text = new Text(name);
        text.setFont(new Font(text.getFont().getFamily(), 5));
        text.scaleXProperty().bind(new
SimpleDoubleProperty(5).divide(scaleProperty()));
        text.scaleYProperty().bind(new
SimpleDoubleProperty(5).divide(scaleProperty()));
        text.setBoundsType(TextBoundsType.VISUAL);
        text.xProperty().bind(circle.centerXProperty().subtract(1));
        text.yProperty().bind(circle.centerYProperty().add(1));

        text.setOnMousePressed(event -> {
            Event.fireEvent(circle, event.copyFor(circle,
event.getTarget()));
            event.consume();
        });
        text.setOnMouseDragged(event -> {
            Event.fireEvent(circle, event.copyFor(circle,
event.getTarget()));
            event.consume();
        });
        text.setOnMouseClicked(event -> {
            Event.fireEvent(circle, event.copyFor(circle,
event.getTarget()));
            event.consume();
        });

        verticesTextsMap.put(name, text);
    }

    private void addEdge(String start, String finish, int weight) {

        Line line = new Line();

line.startXProperty().bind(verticesMap.get(start).centerXProperty());

line.startYProperty().bind(verticesMap.get(start).centerYProperty());

line.endXProperty().bind(verticesMap.get(finish).centerXProperty());

line.endYProperty().bind(verticesMap.get(finish).centerYProperty());
        line.setStrokeWidth(DEFAULT_EDGE_STROKE_WIDTH);
        line.strokeWidthProperty().bind(new
SimpleDoubleProperty(DEFAULT_EDGE_STROKE_WIDTH).divide(scaleProperty()));

        line.setOnMouseClicked(lineEvents.onMouseClickedEventHandler);

        Pair<String,String> startFinish = new Pair<>(start, finish);

        edgesMap.put(startFinish, line);
        reversedEdgesMap.put(line, startFinish);
```

```
        Text text = new Text(weight + "");
        text.setFont(new Font(text.getFont().getFamily(), 4.5));
        text.scaleXProperty().bind(new
SimpleDoubleProperty(4.5).divide(scaleProperty()));
        text.scaleYProperty().bind(new
SimpleDoubleProperty(4.5).divide(scaleProperty()));
        text.setBoundsType(TextBoundsType.VISUAL);

text.xProperty().bind(line.startXProperty().add(line.endXProperty()).divi
de(2).add(2));

text.yProperty().bind(line.startYProperty().add(line.endYProperty()).divi
de(2).add(2));

        text.setOnMouseClicked(event -> {
            Event.fireEvent(line, event.copyFor(line,
event.getTarget()));
            event.consume();
        });

        edgesTextsMap.put(startFinish, text);
    }


    public void notifyColorsChanged() {
        for (Map.Entry<String,Circle> vertex : verticesMap.entrySet()) {
            vertex.getValue().setFill(

vertex.getKey().equals(circleEvents.getSelectedCircleName())
                            ? CircleEvents.SELECTED_VERTEX_COLOR
                            :
getVertexColorByInt(logic.getVertexInfo(vertex.getKey()).color));
        }
        for (Map.Entry<Pair<String,String>,Line> edge :
edgesMap.entrySet()) {
            LogicInterface.EdgeInfo edgeInfo =
logic.getEdgeInfo(edge.getKey().getKey(), edge.getKey().getValue());
            edge.getValue().setStroke(
                    specialColorEdgesContains(edgeInfo)
                            ? SPECIAL_EDGE_COLOR
                            : getEdgeColorByInt(edgeInfo.color));
        }
    }


    public Set<LogicInterface.EdgeInfo> getSpecialColorEdges() {
        return specialColorEdges;
    }
    public void addSpecialColorEdge(LogicInterface.EdgeInfo edge) {
        specialColorEdges.add(edge);
    }
    public void clearSpecialColorEdges() {
        specialColorEdges.clear();
    }
    public boolean specialColorEdgesContains(LogicInterface.EdgeInfo
edge) {
        if (specialColorEdges.contains(edge))
```

```
            return true;
        LogicInterface.EdgeInfo copy = new
LogicInterface.EdgeInfo(edge.finish, edge.start, edge.weight,
edge.color);
        return specialColorEdges.contains(copy);
    }


    public GraphMode getGraphMode() {
        return graphMode;
    }
    public void setGraphMode(GraphMode graphMode) {
        this.graphMode = graphMode;
    }
}
```

файл SummerPracticeApplication:

```
package com.leti.summer_practice.gui.prog;

import com.leti.summer_practice.R;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;


public class SummerPracticeApplication extends Application {


    private static SummerPracticeApplication application;

    private Stage primaryStage;


    public static void launchSummerPracticeApplication(String... args) {
        launch(args);
    }


    @Override
    public void start(Stage primaryStage) throws Exception {
        application = this;
        setPrimaryStage(primaryStage);
        primaryStage.setTitle(R.string("AppName"));
        primaryStage.setScene(new Scene(R.loadFXML("summer-
practice.fxml"), 800, 600));
        primaryStage.show();
    }


    public static SummerPracticeApplication getApplication() {
        return application;
    }
```

```
    public Stage getPrimaryStage() {
        return primaryStage;
    }
    public void setPrimaryStage(Stage primaryStage) {
        this.primaryStage = primaryStage;
    }
}
```

файл SummerPracticeController:

```
package com.leti.summer_practice.gui.prog;

import com.leti.summer_practice.R;
import com.leti.summer_practice.logic.Logic;
import com.leti.summer_practice.logic.LogicInterface;
import com.leti.summer_practice.util.SingleTaskTimer;
import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.*;
import javafx.scene.input.KeyCode;
import javafx.stage.FileChooser;

import java.io.File;
import java.net.URL;
import java.util.Locale;
import java.util.ResourceBundle;
import java.util.TimerTask;


public class SummerPracticeController implements Initializable {


    private static final double STEP_SPEED_CHANGE = 0.1;
    private static final double MIN_STEP_SPEED = 0.1;
    private static final double MAX_STEP_SPEED = 10;

    private static final String LOG_DIVIDER = ";\n";


    @FXML
    public GraphCanvas canvas;
    @FXML
    public TextArea logTextArea;

    @FXML
    public Button modeButton;
    @FXML
    public Button loadFromFileButton;
    @FXML
    public Button clearGraphButton;

    @FXML
```

```
    public TextField speedTextField;
    @FXML
    public Button minusSpeedButton;
    @FXML
    public Button plusSpeedButton;

    @FXML
    public Button startStopButton;
    @FXML
    public Button nextStepButton;
    @FXML
    public Button againButton;


    private final LogicInterface logic = new Logic();

    private boolean answerAlreadyPrinted = false;


    SingleTaskTimer autoStepTimer = new SingleTaskTimer(true);


    @Override
    public void initialize(URL location, ResourceBundle resources) {

        Runnable onSpeedTextFieldUpdated = () -> {

            double period = Double.parseDouble(speedTextField.getText());
            if (period < MIN_STEP_SPEED)
                speedTextField.setText(MIN_STEP_SPEED + "");
            else if (period > MAX_STEP_SPEED)
                speedTextField.setText(MAX_STEP_SPEED + "");

            changeTimerStepSpeed();
        };

        speedTextField.focusedProperty().addListener(observable ->
onSpeedTextFieldUpdated.run());
        speedTextField.setOnKeyPressed(event -> {
            if (event.getCode() == KeyCode.ENTER) {
                onSpeedTextFieldUpdated.run();
                event.consume();
            }
        });

        speedTextField.textProperty().addListener(new ChangeListener<>()
{
            private static final int PRECISION = 2;
            @Override
            public void changed(ObservableValue<? extends String>
observable, String oldValue, String newValue) {
                speedTextField.setText(
                        String.format(
                                new Locale("en_EN"),
                                "%." + PRECISION + "f",
                                Double.parseDouble(newValue)
                        ).replaceAll("([.,]\\d+?)0*$", "$1")
                );
```

```
            }
        });


        canvas.setCameraXY(-canvas.getPrefWidth() / 2, -
canvas.getPrefHeight() / 2);
        canvas.setScale(
                (canvas.getPrefWidth() + canvas.getPrefHeight())
                        / (GraphCanvas.DEFAULT_CANVAS_WIDTH +
GraphCanvas.DEFAULT_CANVAS_HEIGHT));
//        canvas.setCameraXY(
//                GraphCanvas.DEFAULT_CANVAS_CAMERA_X *
canvas.getScale(),
//                GraphCanvas.DEFAULT_CANVAS_CAMERA_Y *
canvas.getScale());

        canvas.setLogic(logic);

        canvas.redraw();
    }


    @FXML
    public void onClearGraphClicked(ActionEvent actionEvent) {

        class ClearGraphAlertContainer {
            public static final ClearGraphAlert CLEAR_GRAPH_ALERT = new
ClearGraphAlert();

            static class ClearGraphAlert extends Alert {
                public ClearGraphAlert() {
                    super(Alert.AlertType.CONFIRMATION,

R.string("clear_graph_confirmation_alert_content_text"),
                            ButtonType.NO,
                            ButtonType.YES);
                    setTitle(R.string("confirmation_alert_title"));
                    setHeaderText(null);
                    ((Button)
getDialogPane().lookupButton(ButtonType.YES)).setDefaultButton(false);
                    ((Button)
getDialogPane().lookupButton(ButtonType.NO)).setDefaultButton(true);
                    ((Button)
getDialogPane().lookupButton(ButtonType.NO)).setCancelButton(true);
                }
            }
        }

        ClearGraphAlertContainer.CLEAR_GRAPH_ALERT.showAndWait();

        if (ClearGraphAlertContainer.CLEAR_GRAPH_ALERT.getResult() ==
ButtonType.YES) {
            logic.clearGraph();
            logTextArea.clear();
            canvas.clearSpecialColorEdges();
            canvas.setLogic(logic);
            canvas.redraw();
```

```
        }
    }


    @FXML
    public void onLoadFromFileClick(ActionEvent actionEvent) {

        class LastDirectoryContainer {
            public static File lastDirectory = null;
        }

        FileChooser fileChooser = new FileChooser();
        if (LastDirectoryContainer.lastDirectory != null)

fileChooser.setInitialDirectory(LastDirectoryContainer.lastDirectory);
        File file =
fileChooser.showOpenDialog(SummerPracticeApplication.getApplication().get
PrimaryStage());
        if (file == null)
            return;
        LastDirectoryContainer.lastDirectory = file.getParentFile();
        try {
            logic.loadFile(file);
        } catch (RuntimeException re) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle(R.string("failed_to_load_file_alert_title"));
            alert.setHeaderText(R.string("error_alert_header"));
            alert.setContentText(re.getMessage());
            alert.showAndWait();
            return;
        }
        canvas.setLogic(logic);
        logTextArea.clear();
        answerAlreadyPrinted = false;
        canvas.clearSpecialColorEdges();
        canvas.redraw();
    }


    @FXML
    public void onStartClick(ActionEvent actionEvent) {

        if (logic.isGraphEmpty())
            return;

        setMovingMode();

        if (!autoStepTimer.hasCurrentTimerTask()) {
            runTimer();
            startStopButton.setText(R.string("stop"));
        } else {
            resetStartStopButtonAndTimer();
        }
    }

    private void resetStartStopButtonAndTimer() {
        autoStepTimer.cancel();
```

```
        startStopButton.setText(R.string("start"));
    }


    @FXML
    public void onNextStepClicked(ActionEvent actionEvent) {

        boolean autoSolvingWasRunning =
autoStepTimer.hasCurrentTimerTask();

        if (autoSolvingWasRunning)
            autoStepTimer.cancel();

        nextStep();

        if (autoSolvingWasRunning)
            runTimerWithDelay();
    }

    private void nextStep() {

        if (logic.isGraphEmpty())
            return;

        setMovingMode();

        if (!logic.isAlgorithmStarted()) {
            boolean algorithmStarted = tryToStartAlgorithm();
            if (!algorithmStarted) {
                autoStepTimer.cancel();
                return;
            }

logTextArea.appendText(R.string("initial_components_vertices_colored") +
LOG_DIVIDER);
            canvas.redraw();
            return;
        }

        if (logic.isAlgorithmFinished()) {
            if (!answerAlreadyPrinted)
                logTextArea.appendText(R.string("connecting_components")
+ LOG_DIVIDER);
            printAlgorithmResult();
            canvas.clearSpecialColorEdges();
            canvas.redraw();
            return;
        }

        LogicInterface.EdgeInfo newEdge;

        while (true) {
            newEdge = logic.getNewEdge();
            if (newEdge == null)
                break;
            if (canvas.specialColorEdgesContains(newEdge)) {
                logTextArea.appendText(
```

```
                        R.string("added_edge")
                                + " "
                                + newEdge.toStringWithoutWeight()
                                + " "
                                + R.string("of_color")
                                + "'"
                                +
GraphCanvas.getEdgeColorByInt(newEdge.color).toString()
                                + "'"
                                + " _ "
                                + R.string("already_added")
                                + LOG_DIVIDER
                    );
            } else {
                break;
            }
        }

        if (newEdge == null) {

            logic.nextBigStep();

            canvas.getSpecialColorEdges().clear();

            logTextArea.appendText(R.string("connecting_components") +
LOG_DIVIDER);

            if (logic.isAlgorithmFinished())
                printAlgorithmResult();

        } else {

            canvas.addSpecialColorEdge(newEdge);

            logTextArea.appendText(
                    R.string("added_edge")
                            + " "
                            + newEdge.toStringWithoutWeight()
                            + " "
                            + R.string("of_color")
                            + "'"
                            +
GraphCanvas.getEdgeColorByInt(newEdge.color).toString()
                            + "'"
                            + LOG_DIVIDER);
        }

        canvas.redraw();
    }

    private boolean tryToStartAlgorithm() {
        try {
            logic.startAlgorithm();
            return true;
        } catch (RuntimeException re) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
```

```
alert.setTitle(R.string("failed_to_start_algorithm_alert_title"));
            alert.setHeaderText(R.string("error_alert_header"));
            alert.setContentText(re.getMessage());
            alert.showAndWait();
            return false;
        }
    }

    private void printAlgorithmResult() {

        if (logic.isGraphEmpty() || !logic.isAlgorithmFinished())
            return;

        if (answerAlreadyPrinted)
            return;
        answerAlreadyPrinted = true;

        StringBuilder sb = new StringBuilder();

        if (!logTextArea.getText().isEmpty())
            sb.append('\n');

        sb.append(R.string("algorithm_result_header")).append('\n');

        for (LogicInterface.EdgeInfo edge : logic.getAnswer())
            sb.append(edge.toStringWithoutWeight()).append(" =
").append(edge.weight).append('\n');

        if (sb.charAt(sb.length() - 1) == '\n')
            sb.setCharAt(sb.length() - 1, '.');

        logTextArea.appendText(sb.toString());
    }


    @FXML
    public void onAgainClicked(ActionEvent actionEvent) {
        resetStartStopButtonAndTimer();
        logTextArea.clear();
        answerAlreadyPrinted = false;
        logic.killAlgorithm();
        canvas.clearSpecialColorEdges();
        canvas.redraw();
    }


    @FXML
    public void onModeClick(ActionEvent actionEvent) {
        if (canvas.getGraphMode() == GraphCanvas.GraphMode.MOVING)
            setDrawingMode();
        else if (canvas.getGraphMode() == GraphCanvas.GraphMode.DRAWING)
            setMovingMode();
        else
            throw new Error("Stub!");
    }
```

```
    private void setDrawingMode() {
        canvas.setGraphMode(GraphCanvas.GraphMode.DRAWING);
        logic.killAlgorithm();
        canvas.clearSpecialColorEdges();
        canvas.redraw();
        modeButton.setText(R.string("drag_mode"));
    }
    private void setMovingMode() {
        canvas.setGraphMode(GraphCanvas.GraphMode.MOVING);
        modeButton.setText(R.string("draw_mode"));
    }


    @FXML
    public void onMinusSpeedClick(ActionEvent actionEvent) {
        double oldValue = Double.parseDouble(speedTextField.getText());
        double newValue = oldValue - STEP_SPEED_CHANGE;
        if (newValue < MIN_STEP_SPEED)
            return;
        speedTextField.setText(String.valueOf(newValue));

        changeTimerStepSpeed();
    }

    @FXML
    public void onPlusSpeedClick(ActionEvent actionEvent) {
        double oldValue = Double.parseDouble(speedTextField.getText());
        double newValue = oldValue + STEP_SPEED_CHANGE;
        if (newValue > MAX_STEP_SPEED)
            return;
        speedTextField.setText(String.valueOf(newValue));

        changeTimerStepSpeed();
    }


    private void changeTimerStepSpeed() {
        if (!autoStepTimer.hasCurrentTimerTask())
            return;
        autoStepTimer.cancel();
        long period = (long)(1000 /
Double.parseDouble(speedTextField.getText()));
        long time = autoStepTimer.timeFromLastCompletion();
        long delay = period - time;
        if (delay < 0)
            delay = 0;
        runTimer(delay, period);
    }

    private void runTimer() {
        runTimer(0, (long)(1000 /
Double.parseDouble(speedTextField.getText())));
    }

    private void runTimer(long delay, long period) {
        autoStepTimer.schedule(new TimerTask() {
            @Override
            public void run() {
```

```
                Platform.runLater(() -> {
                    nextStep();
                    if (logic.isAlgorithmFinished())
                        autoStepTimer.cancel();
                });
            }
        }, delay, period);
    }

    private void runTimerWithDelay() {
        long delay_period = (long)(1000 /
Double.parseDouble(speedTextField.getText()));
        runTimer(delay_period, delay_period);
    }
}
```