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

ОТЧЕТ

по учебной практике

Тема: Алгоритм Дейкстры поиска кратчайших путей в графе

Студент гр. 9382 Павлов Р.В.

Студент гр. 9382 Рыжих Р.В.

Студентка гр. 9382 Сорочина М.В.

Руководитель Фирсов М.А.

Санкт-Петербург
2021

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Павлов Р.В. группы 9382

Студент Рыжих Р.В. группы 9382

Студентка Сорочина М.В. группы 9382

Тема практики: Алгоритм Дейкстры поиска кратчайших путей в графе

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: Дейкстры

Сроки прохождения практики: 01.07.2021 – 14.07.2021

Дата сдачи отчета: 12.07.2021

Дата защиты отчета: 13.07.2021

Студент		Павлов Р.В.
Студент		Рыжих Р.В.
Студентка		Сорочина М.В.
Руководитель		Фирсов М.А.

АННОТАЦИЯ

Целью практической работы является разработка приложения, в котором реализуется алгоритм Дейкстры с визуализацией. Для выполнения задачи используется язык программирования Java. Пользователю предоставляется возможность нарисовать граф или загрузить из файла и запустить автоматическое или пошаговое выполнение алгоритма поиска кратчайших путей в графе. Приложение должно быть понятным и удобным для использования. Задание выполняется командой из трех человек, за каждым закреплены определенные роли.

SUMMARY

The purpose of the practical work is to develop an application that implements Dijkstra's algorithm with visualization. The Java programming language is used to complete the task. The user is given the opportunity to draw a graph or load from a file and start an automatic or step-by-step execution of the algorithm for finding the shortest paths in the graph. The application should be clear and easy to use. The task is carried out by a team of three people, each assigned to specific roles.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью учебной практики является создание приложения, визуализирующего работу алгоритма Дейкстры, предназначенного для нахождения всех кратчайших путей взвешенного графа с неотрицательными весами из исходной вершины до каждой вершины графа, последовательно наращивая множество вершин, для которых известен кратчайший путь. Приложение должно быть написано на языке Java и снабжено понятным и удобным в использовании графическим интерфейсом. Пользователю должна быть предоставлена возможность ввести исходные данные в самой программе с использованием клавиатуры и мыши. Результат работы алгоритма также должен выводиться на экран. Должна быть предоставлена возможность моментального отображения результата.

Задание выполняется командой из трех человек, за каждым из которых закреплены определенные обязанности – реализация графического интерфейса, логики алгоритма, проведение тестирования, сборка проекта и написание отчета. Готовая программа должна корректно собираться из исходников в один исполняемый jar-архив.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

Начало работы.

Пользователю доступен начальный экран (до запуска алгоритма), на начальном экране можно:

- Нарисовать граф (добавление вершин (двойной клик ЛКМ), протягивание рёбер между ними ("Shift"+перетаскивание курсора на другую вершину, удерживая ЛКМ), указание весов рёбер (клик ЛКМ по ребру, указание в меню на правой панели веса ребра), удаление вершин и рёбер (клик ЛКМ по вершине или ребру, нажатие кнопки "Удалить" в правом меню)
- Удалить граф (полностью стирается нарисованный граф)
- Сохранить нарисованный граф (число вершин, их координаты (пары <x:y>), затем идут разделённые специальным символом "/" наборы пар вида <вершина:вес_ребра> (может быть несколько пар в одной последовательности), являющиеся списками смежности вершин, номера которых соответствуют порядковым номерам наборов в файле)
- Загрузить граф из файла, отрисовать его (загрузить из сохранённого ранее файла, к примеру)
- Начать выполнение программы (кнопка "Начать")

Подготовка к выполнению алгоритма.

После нажатия кнопки "Начать" пользователю предлагается ввести некоторые начальные данные для работы алгоритма:

- Временной интервал, который определяет частоту выполнения шагов алгоритма и отрисовки в автоматическом режиме (см. далее)

- Начальную вершину. Выбрать её можно, кликнув по какой-либо вершине (можно выбрать несколько раз). Затем необходимо подтвердить выбор нажатием кнопки "Подтвердить"

Выполнение алгоритма.

Алгоритм начинает работу, рассматривая начальную вершину. Пользователю доступен выбор режима: автоматический или пошаговый (изначально по умолчанию включён пошаговый).

- Автоматический: включается кнопкой "Авто", каждый шаг выполняется через указанный временной интервал, кнопка перехода к след. шагу "Шаг" неактивна, можно приостановить автоматический режим, нажав на кнопку "Авто" во время выполнения
- Пошаговый: при условии, что автоматический режим отключён, кнопка перехода к следующему шагу "Шаг" активна и приводит к его выполнению

В процессе работы алгоритма можно включать/отключать автоматический режим, если отключён - последовательно выполнять шаги в пошаговом режиме. Также можно нажать кнопку "Сброс", которая остановит выполнение алгоритма, удалит все промежуточные данные и вернёт программу в режим "Начало работы", сохранив при этом нарисованный граф и вернув начальную раскраску (все вершины помечены одним цветом 3, см. далее)

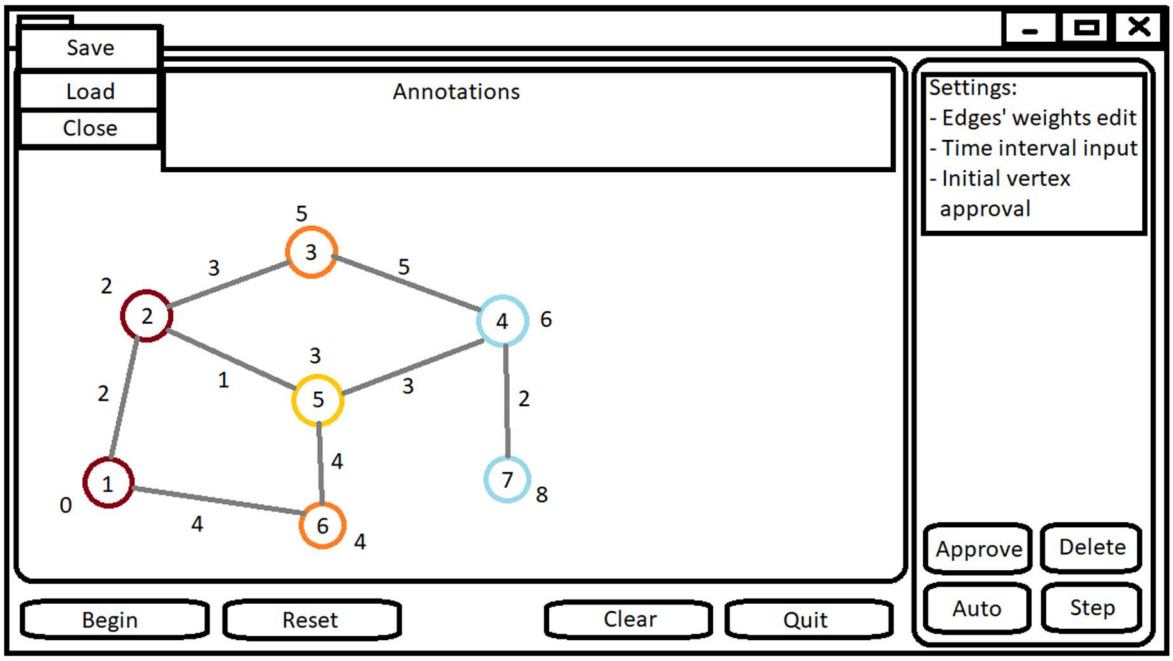
На каждом шаге алгоритма:

- Отработанные вершины (до которых путь уже найден) помечаются цветом 1
- Вершины в очереди с приоритетом, которые подлежат рассмотрению, помечаются цветом 2
- Вершины, до которых ещё не найден хоть какой-либо (пусть не самый короткий) путь, помечаются цветом 3

- Текущая рассматриваемая вершина помечается цветом 4. Путь до этой вершины найден на данном шаге и указывается рядом с ней в виде подписи, о данном событии сообщается пользователю, сообщение имеет форму краткой заметки в специальном поле.

По завершении работы алгоритма программа выводит итоги работы алгоритма (например, вершина, длина пути до неё, последовательность вершин, являющихся этим путём - так для каждой вершины) и возвращается в режим "Начало работы".

Предполагаемый вид интерфейса.



1.2. Уточнение требований после сдачи прототипа

- Заголовок окна — название алгоритма;
- Увеличить высоту поля «Аннотации»;
- Поле «Установки» не должно быть единым. Для каждой настройки должен быть свой элемент управления;
- Текст в «Аннотациях» должен быть выделяемым и копируемым;
- В меню File «Загрузить» должно быть над «Сохранить».

1.3. Уточнение требований после сдачи 1-ой версии

- Сделать поле «Аннотации» изменяемого размера (чтобы можно было увеличивать его высоту, уменьшая высоту области рисования графа) — в 3-ей версии;
- При выделении ребра справа должны появляться слова «Задать вес»;
- Увеличить размер шрифта для номеров вершин на рисунке;
- Уточнить формат представления графа в файле;
- Во 2-ой версии, кроме визуализации, должны выводиться и пояснения в ходе работы алгоритма

1.4. Уточнение требований после сдачи 2-ой версии

- Запретить пользоваться пунктами «Загрузить» и «Сохранить» после начала выполнения алгоритма;
- Кнопка «Шаг» не должна быть активна до начала выполнения;
- Вершины, до которых уже определено минимальное расстояние, сделать немного светлее (черный текст на синем фоне плохо виден);
- На каждом шаге подсвечивать найденный путь, выделяя ребра, входящие в него, другим цветом

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

- Прототип (07.07.2021): Разработка GUI, позволяющего пользователю создавать объекты на поле без привязки к структуре данных (графу).
- 1-я версия (09.07.2021): Разработка алгоритма Дейкстры, реализация классов "Vertex" и "Solver". "Vertex" – класс вершины, содержащий необходимые данные и методы для работы с каждой вершиной, "Solver" – класс, содержащий граф и методы для взаимодействия с ним, а также реализацию алгоритма Дейкстры. Реализация автоматического выполнения алгоритма после нажатия кнопки "Begin" с выводом сообщений о его работе в поле "Annotations" и загрузки входных данных из файла (кнопка "Load")
- 2-я версия (12.07.2021): Добавление визуализации работы алгоритма на каждом шаге и режимов работы "Пошаговый"/"Автоматический".
- 3-я версия (14.07.2021): Расширение функционала: добавление функций сохранения (кнопка "Save"), сброса к начальному состоянию графа в ходе работы алгоритма (кнопка "Reset") подтверждения выбора начальной вершины (кнопка "Approve"), задания весов рёбер и временного интервала в отдельном поле (меню справа). Тестирование программы выполнением алгоритма на различных входных данных, а также использованием доступного функционала GUI. Исправление ошибок. Оформление отчёта.

2.2. Распределение ролей в бригаде

Павлов Роман: создание структур данных и их методов, необходимых для реализации алгоритма Дейкстры; реализация алгоритма Дейкстры; привязка GUI к методам взаимодействия со структурами данных.

Рыжих Роман: разработка GUI (рисование вершин и ребер графа), дизайн интерфейса, привязка GUI к методам взаимодействия со структурами данных.

Сорочина Мария: разработка GUI (логика работы элементов окна, не относящихся непосредственно к рисованию), тестирование алгоритма и GUI, оформление и структурирование документации.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

- Класс Window – класс окна приложения.
    - private AutoMode thr - поток автоматического режима
    - private final Solver solver - объект, выполняющий алгоритм
    - private final CustomLogger logger - объект, предоставляющий записанные данные
    - private final JSplitPane splitPane - разделенная панель с регулировкой высоты компонентов
    - private final JPanel annotationsPanel - панель для вывода данных
    - private final GPanel canvasPanel - панель для отображения графа
    - private final JPanel settingsPanel - панель установок и взаимодействия с объектами графа
    - private final JPanel bottomPanel - нижняя панель, содержащая кнопки "Начать", "Сброс", "Очистить", "Закрыть"
    - private final JMenuItem loadButton - кнопка загрузки графа из файла
    - private final JMenuItem saveButton - кнопка сохранения графа в файл
    - private final JTextArea textArea - область вывода промежуточных и итоговых данных
    - private final JLabel infoLabel - метка с информацией о возможных действиях
    - private final JTextField textField - текстовое поле для ввода данных
    - private final JButton approveButton - кнопка подтверждения выбора начальной вершины

    - private final JButton setTimeButton - кнопка задания интервала между шагами автоматического режима
    - private final JButton setButton - кнопка установки веса ребра
    - private final JButton deleteButton - кнопка удаления вершины или ребра
    - private final JToggleButton autoButton - кнопка запуска автоматического режима
    - private final JButton stepButton - кнопка пошагового режима
    - private final JButton beginButton - кнопка, запускающая инициализацию алгоритма
    - private final JButton resetButton - кнопка сброса в режим редактирования
    - private final JButton clearButton - кнопка очистки панели с графом
    - private final JButton closeButton - кнопка закрытия приложения
- AutoMode - класс потока автоматического режима
    - private final GPanel canvas - панель с графом
    - private final JTextArea textArea - область для записи данных
    - private final JToggleButton autoButton - кнопка автоматического режима
    - private final JButton stepButton - кнопка пошагового режима
    - private final Solver solver - объект, выполняющий алгоритм
    - private final CustomLogger logger - объект, предоставляющий записанные данные
    - private final int period - временной интервал шага для автоматического выполнения
    - private boolean active - режим работы/ожидания потока
    - private boolean alive - продолжение/завершение работы потока
- Solver - класс-исполнитель алгоритма
    - private Vertex init - начальная вершина

    - private Vertex prev - предыдущая рассмотренная вершина
    - private final ArrayList<Vertex> vertSet - массив вершин графа
    - private final PriorityQueue<Vertex> front - достижимые на данный момент вершины
- CustomLogger - класс для записи данных во время выполнения алгоритма, а также для последующего вывода записанных данных
    - private final ArrayList<String> messages - массив хранимых сообщений
    - private int nextMessageIndex - индекс следующего сообщения
    - private boolean endReached - флаг достижения конца записанных сообщений
- Vertex - класс, хранящий информацию о вершине
    - private int id - номер вершины
    - private int pathLen - длина пути от начальной вершины
    - private Vertex parent - предыдущая вершина, используется при записи кратчайшего пути
    - private final HashMap<Vertex, Integer> adjList - список смежных вершин
    - private Colors color - цвет вершины
- VisualVertex - класс, хранящий информацию о вершинах для их отрисовки
    - private int x - координата по x
    - private int y - координата по y
    - private int id - номер вершины
- VisualEdge - класс, хранящий информацию о ребрах для их отрисовки
    - private VisualVertex v1 и private VisualVertex v2 - вершины, соединенные данным ребром
    - private Shape line - фигура для отображения ребра
    - private int weight - вес ребра

- FileHandler - класс для обработки данных из файла
    - private File file - экземпляр класса File для записи пути до файла и его загрузки
- GPanel - класс-наследник JPanel с возможностью рисования.
    - private final Font FONT - шрифт надписей на вершинах
    - private final int RADIUS - радиус вершин
    - private final Solver solver - объект, выполняющий алгоритм
    - private final ArrayList<VisualVertex> vertices - список вершин
    - private final ArrayList<VisualEdge> edges - список ребер
    - private VisualVertex chosenVertex - выбранная вершина
    - private VisualVertex initVertex - начальная вершина
    - private VisualVertex lastConsideredVertex - последняя рассмотренная вершина
    - private VisualVertex draggedVertex - перетаскиваемая вершина
    - private VisualEdge chosenEdge - выбранное ребро
    - private VisualEdge drawnEdge - рисуемое ребро
    - boolean drawingEdge - флаг режима рисования ребра
    - boolean holdingVertex - true, если вершина зажата, false - в ином случае
    - boolean draggingVertex - флаг режима перетаскивания вершин
    - private boolean isEditable - флаг режима редактирования графа, true - если режим редактирования доступен
    - private boolean choosingInit - флаг режима выбора начальной вершины, true, если режим доступен
- Перечисление Colors - 4 цвета для выделения вершин во время работы алгоритма
    - COLOR1 - белый цвет, используется для вершин, которые еще не участвовали в работе алгоритма
    - COLOR2 - розовый цвет, используется для вершины, которая рассматривается на некотором шаге алгоритма

- COLOR3 - оранжевый цвет, используется для вершин, которые будут рассматриваться следующими
- COLOR4 - светло-серый цвет, используется для вершин, которые уже пройдены

3.2. Основные методы

- Window
  - private void init() - инициализация окна, настройка всех панелей и кнопок
  - public void onVertexChoice(int id) - принимает номер вершины (id), выводит сообщение о том, что вершина выбрана и делает активной кнопку удаления вершины
  - public void onEdgeChoice(int weight) - принимает вес ребра (weight), выводит соответствующий текст, дает возможность изменить вес или удалить ребро
- AutoMode
  - public void run() - функция автоматического выполнения шагов алгоритма с заданной задержкой
- Solver
  - public int addVertex() - добавление вершины
  - public void deleteVertex(int id) - принимает номер вершины (id), удаляет из списка вершин
  - public void addEdge(int from, int to, int dist) - принимает два номера вершин (from, to) и вес ребра (dist), добавляет ребро заданного веса между данными вершинами
  - public void setEdgeWeight(int from, int to, int weight) - принимает два номера вершин (from, to) и вес ребра (weight), меняет вес ребра, заданного вершинами, на передаваемый

    - public void deleteEdge(int from, int to) - принимает два номера вершин (from, to), удаляет ребро, соединяющее две передаваемые вершины
    - public void setInit(int init) - принимает номер вершины (init), делает эту вершину стартовой
    - public boolean step(CustomLogger logger) - принимает объект для записи данных, выполняет шаг алгоритма
    - public void clear() - очищает граф
    - public void reset() - сбрасывает граф до начального состояния
    - public ArrayList<String> results() - функция получения конечных результатов
- CustomLogger
    - public void addMessage(String message) - принимает строку с сообщением (message), добавляет полученную строку в список
    - public String getNextMessage() - возвращает строку, в которой содержатся все еще не выведенные сообщения
- Vertex
    - public void addToAdjList(Vertex v, Integer dist) - принимает вершину (v) и расстояние до нее (dist), добавляет передаваемую вершину в список смежности
    - public void setPathLen(int pathLen) - принимает длину пути (pathLen), устанавливает новое значение длины пути для данной вершины
    - public void setParent(Vertex v) - принимает вершину (v), устанавливает вершину v как предыдущую для данной
    - public void setColor(Colors color) - принимает цвет (color), меняет цвет вершины
    - public HashMap<Vertex, Integer> getAdjList() - возвращает список смежных вершин
- FileHandler

    - public void save() - сохраняет граф в файл
    - public ArrayList<Integer> load() - загружает граф из файла
- GPanel
    - public void clear() - удаляет граф
    - public void uncheck() - снимает выбор с вершины или ребра
    - private VisualVertex chooseCircle(int x, int y) - принимает координаты, возвращает вершину, если координаты попадают в нее, иначе возвращает null
    - public void setChoosingInit(boolean flag) - устанавливает режим выбора начальной вершины в зависимости от флага
    - public void setEditable(boolean flag) - в зависимости от флага позволяет/запрещает редактировать граф
    - public boolean start() - запускает работу алгоритма
    - public void finish() - завершает работу алгоритма
    - public void deleteVertex() - удаляет выбранную вершину
    - public void deleteEdge() - удаляет выбранное ребро
    - private void setShape(VisualEdge edgeDrawn) - устанавливает фигуру для отображения ребра
    - public void load() - загрузка из файла
    - public void save() - сохранение в файл
    - public void setEdgeWeight(int w) - установка веса выделенного ребра
    - public void paintComponent(Graphics g2) - отрисовка графа

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

Для тестирования графического интерфейса было запущено приложение и проверена работа всех нарисованных объектов и кнопок. На рис. 1 представлен графически введенный граф с пошаговым запуском алгоритма.

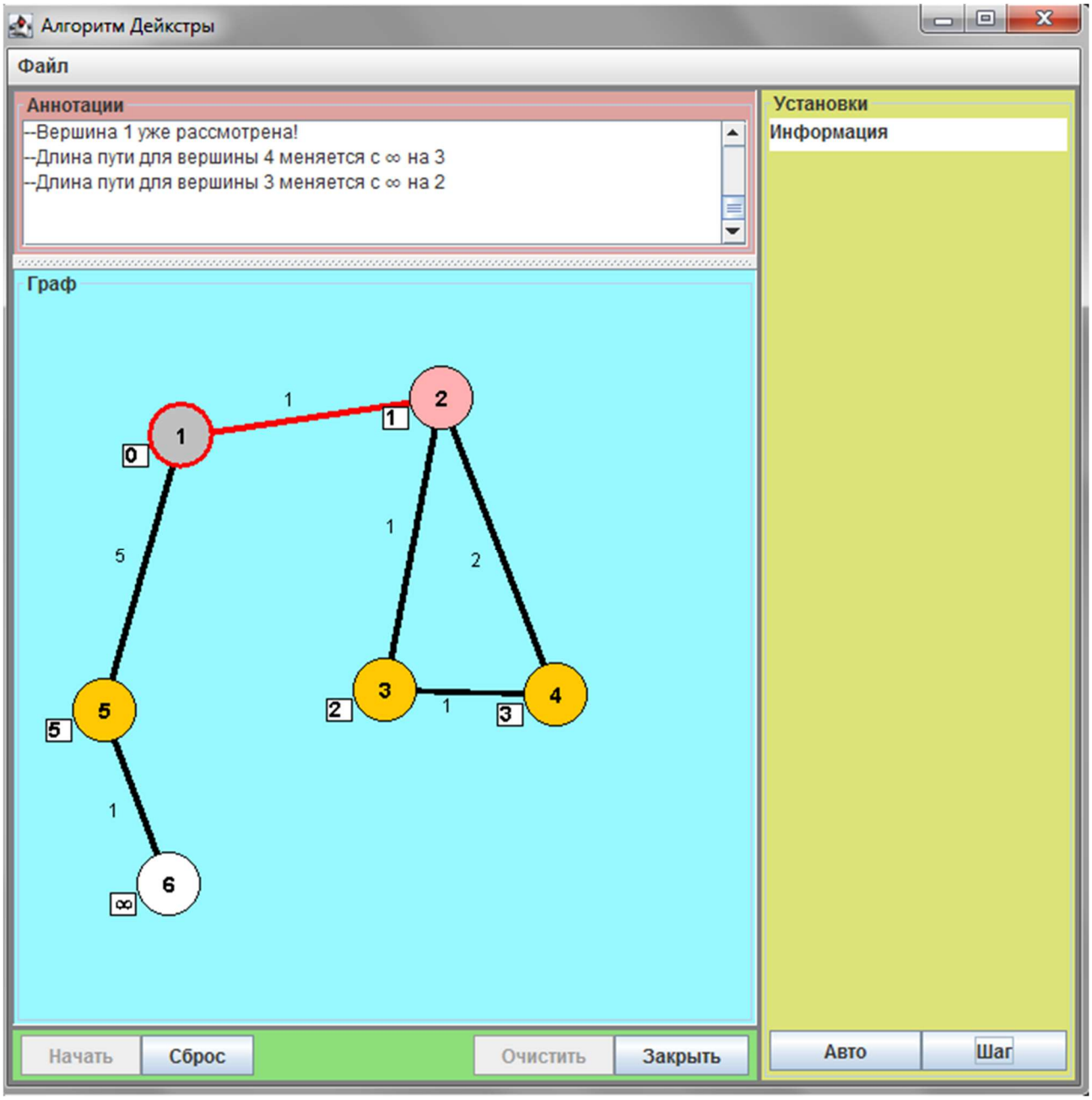


Рис. 1. Графический ввод графа и запуск пошагового выполнения

На рис. 2 представлена работа программы при выборе пункта меню “Сохранить”.

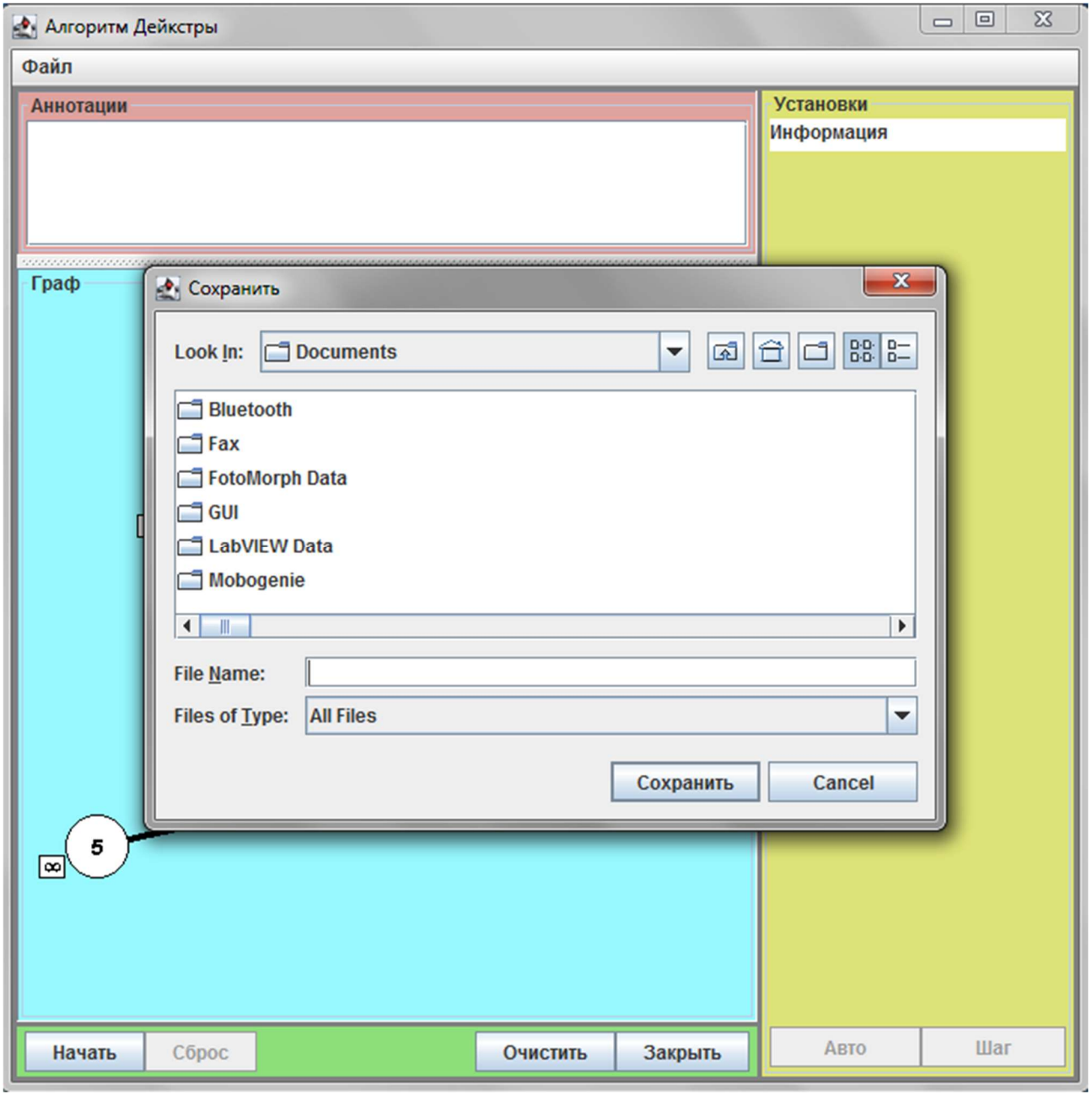


Рис. 2. Сохранение в файл.

На рис. 3 представлена работа программы после выбора пункта “Загрузить” в меню.

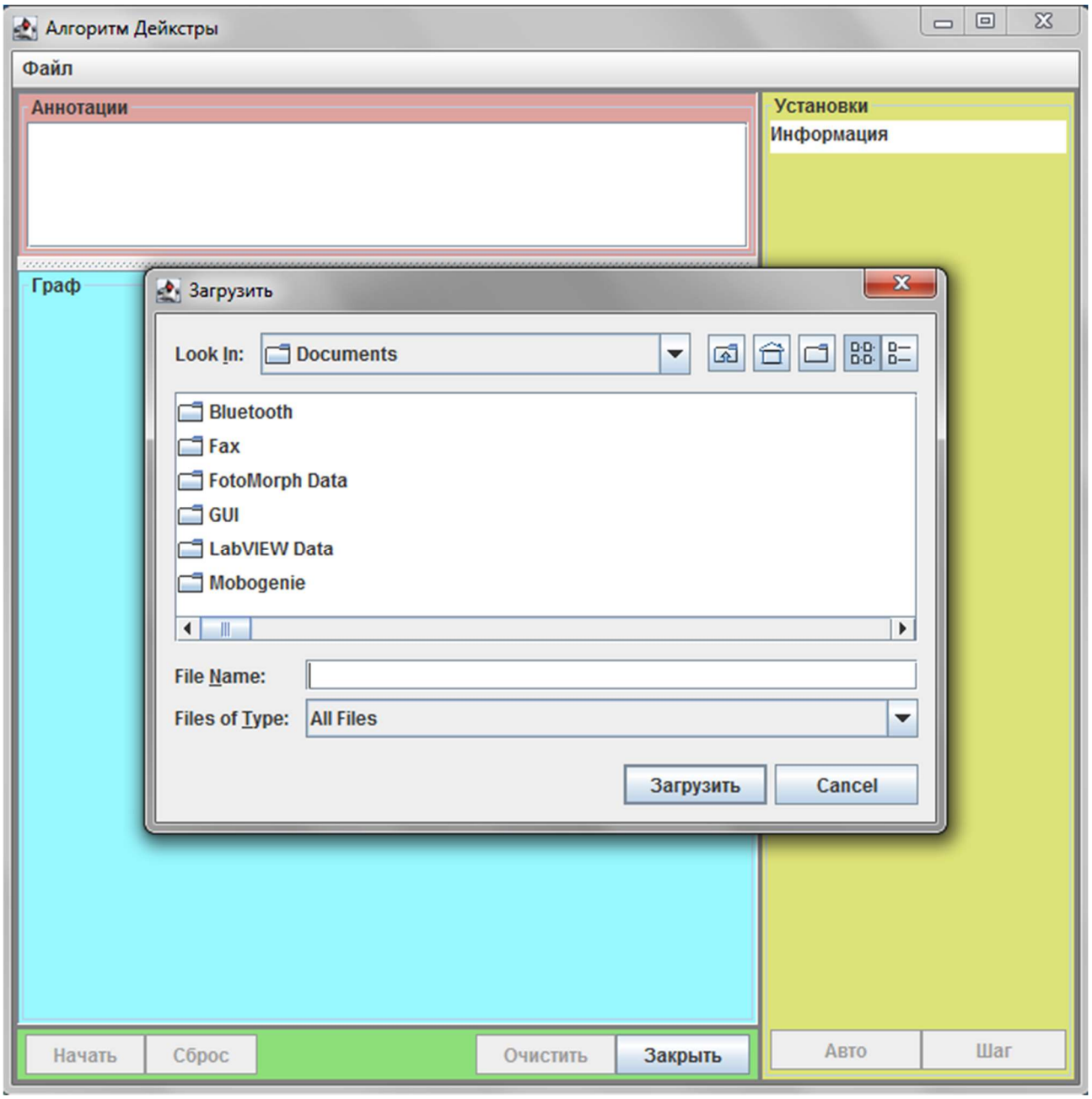


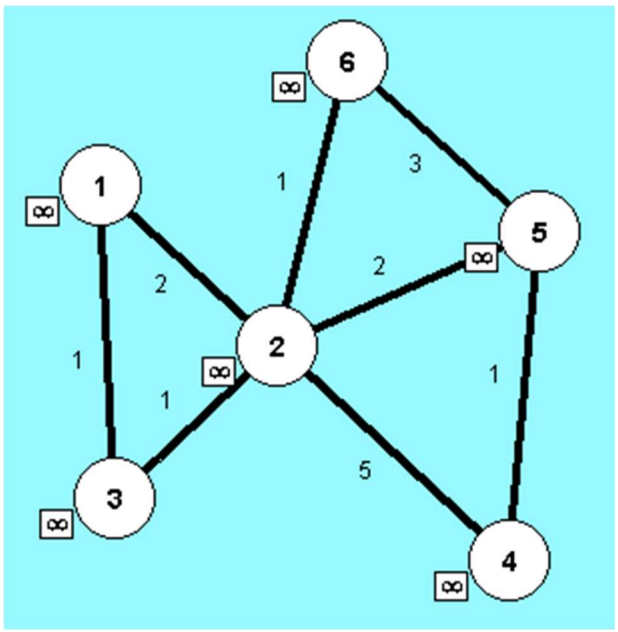
Рис. 3. Загрузка из файла.

Вручную был проверен весь функционал программы.

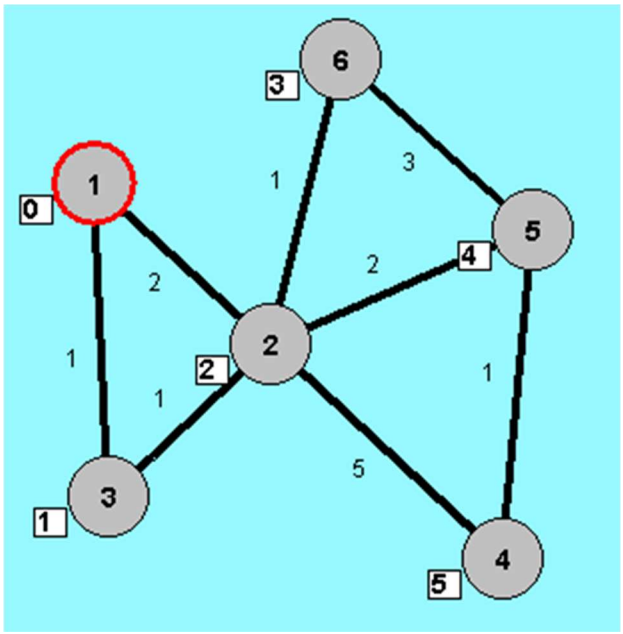
4.2. Тестирование кода алгоритма

Тест 1.

Ввод:



Графический вывод:



Текстовый вывод:

Путь: 1
Длина пути: 0
Рассматриваются смежные вершины:
--Длина пути для вершины 3 меняется с ∞ на 1
--Длина пути для вершины 2 меняется с ∞ на 2

3. Путь: 1 3
Длина пути: 1

Рассматриваются смежные вершины:
--Вершина 1 уже рассмотрена!
--Для вершины 2 длина пути остается прежней

2. Путь: 1 2
Длина пути: 2
Рассматриваются смежные вершины:
--Длина пути для вершины 4 меняется с ∞ на 7
--Вершина 3 уже рассмотрена!
--Длина пути для вершины 5 меняется с ∞ на 4
--Вершина 1 уже рассмотрена!
--Длина пути для вершины 6 меняется с ∞ на 3

6. Путь: 1 2 6
Длина пути: 3
Рассматриваются смежные вершины:
--Для вершины 5 длина пути остается прежней
--Вершина 2 уже рассмотрена!

5. Путь: 1 2 5
Длина пути: 4
Рассматриваются смежные вершины:
--Длина пути для вершины 4 меняется с 7 на 5
--Вершина 6 уже рассмотрена!
--Вершина 2 уже рассмотрена!

4. Путь: 1 2 5 4
Длина пути: 5
Рассматриваются смежные вершины:
--Вершина 5 уже рассмотрена!
--Вершина 2 уже рассмотрена!

Итоги:
1. Путь: 1
Длина пути: 0

2. Путь: 1 2
Длина пути: 2

3. Путь: 1 3
Длина пути: 1

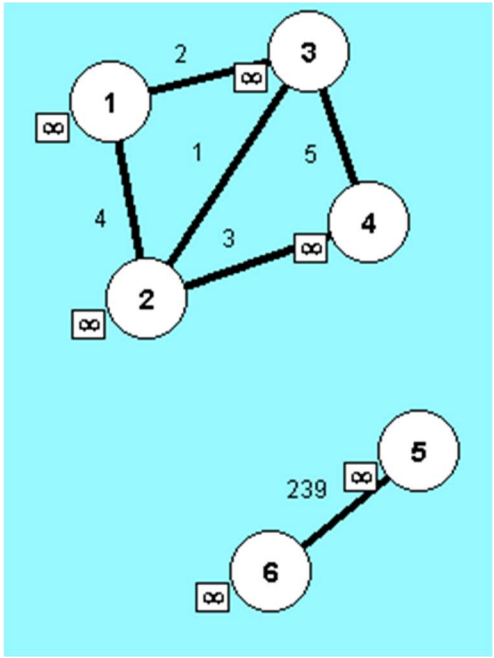
4. Путь: 1 2 5 4
Длина пути: 5

5. Путь: 1 2 5
Длина пути: 4

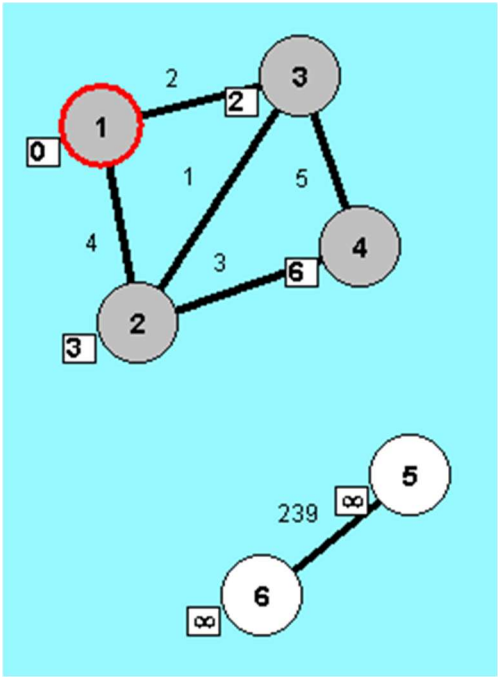
6. Путь: 1 2 6
Длина пути: 3

Тест 2.

Ввод:



Графический вывод:



Текстовый вывод:

1. Путь: 1
Длина пути: 0
Рассматриваются смежные вершины:
--Длина пути для вершины 2 меняется с ∞ на 4
--Длина пути для вершины 3 меняется с ∞ на 2

3. Путь: 1 3
Длина пути: 2
Рассматриваются смежные вершины:
--Вершина 1 уже рассмотрена!
--Длина пути для вершины 4 меняется с ∞ на 7
--Длина пути для вершины 2 меняется с 4 на 3

2. Путь: 1 3 2
Длина пути: 3
Рассматриваются смежные вершины:
--Вершина 1 уже рассмотрена!
--Длина пути для вершины 4 меняется с 7 на 6
--Вершина 3 уже рассмотрена!

4. Путь: 1 3 2 4
Длина пути: 6

Рассматриваются смежные вершины:
--Вершина 2 уже рассмотрена!
--Вершина 3 уже рассмотрена!

Итоги:
1. Путь: 1
Длина пути: 0

2. Путь: 1 3 2
Длина пути: 3

3. Путь: 1 3
Длина пути: 2

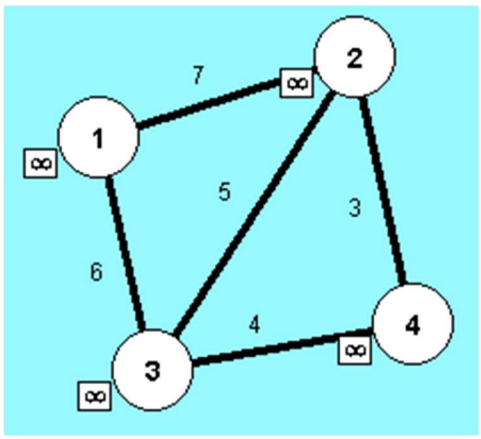
4. Путь: 1 3 2 4
Длина пути: 6

5. Путь: не существует
Длина пути: ∞
6. Путь: не существует
Длина пути: ∞

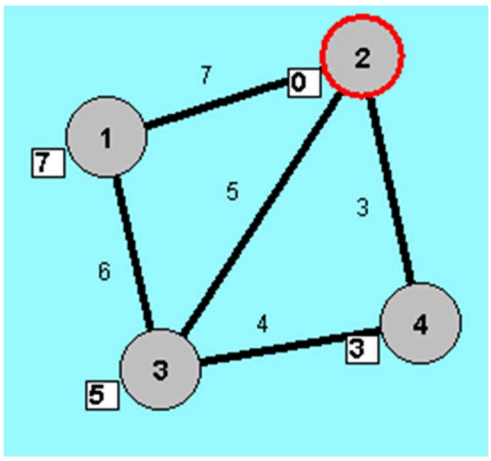
Тест 3.
Ввод: (из файла)

```
4
63 123
191 83
90 239
220 216
3 6
2 7
/
4 3
3 5
/
4 4
```

Графическое представление:



Графический вывод:



Текстовый вывод:

2. Путь: 2
Длина пути: 0
Рассматриваются смежные вершины:
--Длина пути для вершины 4 меняется с ∞ на 3
--Длина пути для вершины 1 меняется с ∞ на 7
--Длина пути для вершины 3 меняется с ∞ на 5

4. Путь: 2 4
Длина пути: 3
Рассматриваются смежные вершины:
--Вершина 2 уже рассмотрена!
--Для вершины 3 длина пути остается прежней

3. Путь: 2 3
Длина пути: 5
Рассматриваются смежные вершины:
--Вершина 4 уже рассмотрена!

--Для вершины 1 длина пути остается прежней
--Вершина 2 уже рассмотрена!

1. Путь: 2 1
Длина пути: 7
Рассматриваются смежные вершины:
--Вершина 3 уже рассмотрена!
--Вершина 2 уже рассмотрена!

Итоги:
1. Путь: 2 1
Длина пути: 7

2. Путь: 2
Длина пути: 0

3. Путь: 2 3
Длина пути: 5

4. Путь: 2 4
Длина пути: 3

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы было разработано приложение, визуализирующее алгоритм Дейкстры, предназначенный для поиска кратчайших путей от начальной вершины до всех остальных. Приложение позволяет построить граф и запустить выполнение алгоритма в пошаговом или автоматическом (в котором можно указать скорость выполнения) режиме. Работа алгоритма сопровождается цветовым выделением вершин и рёбер в графе и указанием длин найденных путей рядом с вершинами, текстовыми пояснениями с промежуточными данными. Итоги работы также выводятся в текстовой форме.

В результате были получены такие навыки, как работа в команде, написание программ на ЯП Java с использованием знаний о строении объектно-ориентированных программ, реализация графического интерфейса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Учебный курс по Java на Stepik // Stepik. URL: https://stepik.org/course/187?auth=registration (дата обращения: 01.07.2021).
2. Шилдт Г. Java. Полное руководство. 2018. 1488 с.
3. Документация Java // Oracle Help Center. URL: https://docs.oracle.com/
4. Алгоритм Дейкстры // Викиконспекты. URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ.

Window.java:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class AutoMode extends Thread{
    private final GPanel canvas;    // Панель с графом
    private final JTextArea textArea;   // Область для записи данных
    private final JToggleButton autoButton; // Кнопка автоматического режима
    private final JButton stepButton;   // Кнопка пошагового режима
    private final Solver solver;    // Объект, выполняющий алгоритм
    private final CustomLogger logger;  // Объект, предоставляющий записанные данные
    private final int period;   // Временной интервал автоматического выполнения
    private boolean active = true;  // Режим работы/ожидания потока
    private boolean alive = true;   // Продолжение/завершение работы потока

    public AutoMode(GPanel canvas, JTextArea ta, JToggleButton ab, JButton sb, Solver sol,
CustomLogger log, int period){
        super();
        this.solver = sol;
        this.logger = log;
        this.textArea = ta;
        this.canvas = canvas;
        this.autoButton = ab;
        this.stepButton = sb;
        this.period = period;
    }

    public void run() { // Выполнение шагов алгоритма с заданной задержкой
        try {
            boolean running = solver.step(logger);
            canvas.getParent().repaint();
            while (running) {
                synchronized (this){
                    if(!active){
                        wait();
                    }
                }
                if(!alive){
                    alive = true;
                    return;
                }
                textArea.append(logger.getNextMessage() +"\n\n");
                textArea.setCaretPosition(textArea.getDocument().getLength());
                try {
                    Thread.sleep(period);
                    synchronized (this){
                        if(!active){
                            wait();
                        }
                    }
                    if(!alive){
```

```
                    alive = true;
                    return;
                }
                running = solver.step(logger);
                canvas.getParent().repaint();
            } catch (InterruptedException e) {
                interrupt();
            }
        }
    }
    catch(InterruptedException e){
        e.printStackTrace();
    }
    autoButton.setSelected(false);
    autoButton.setEnabled(false);
    stepButton.setEnabled(false);
    StringBuilder results = new StringBuilder();
    for(String s : solver.results()) {
        results.append(s).append("\n");
    }
    textArea.append("Итоги:\n" + results);
    textArea.setCaretPosition(textArea.getDocument().getLength());
}

public void disable(){
    this.active = false;
}

public void enable(){
    this.active = true;
    synchronized (this) {
        notify();
    }
}

public void kill(){
    alive = false;
    enable();
}
}

public class Window extends JFrame{
    private AutoMode thr = null; // Поток автоматического режима
    private final Solver solver = new Solver(); // Объект, выполняющий алгоритм
    private final  CustomLogger logger = new CustomLogger(); // Объект, предоставляющий
записанные данные
    private final JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT); //
Разделённая панель с регулировкой высоты компонентов
    private final JPanel annotationsPanel = new JPanel();   // Панель, содержащая область
вывода данных
    private final GPanel canvasPanel = new GPanel(solver);  // Панель для отображения графа
    private final JPanel settingsPanel = new JPanel();  // Панель установок и взаимодействия с
объектами графа
    private final JPanel bottomPanel = new JPanel();    // Нижняя панель с кнопками "Начать",
"Сброс", "Очистить", "Закрыть"
    private final JMenuItem loadButton = new JMenuItem("Загрузить");    // Кнопка загрузки графа
из файла
```

```
    private final JMenuItem saveButton = new JMenuItem("Сохранить");    // Кнопка сохранения графа в файл
    private final JTextArea textArea = new JTextArea(); // Область вывода данных
    private final JLabel infoLabel = new JLabel("Информация");  // Метка с информацией о предлагаемых действиях
    private final JTextField textField = new JTextField();  // Текстовое поле для ввода данных
    private final JButton approveButton = new JButton("Подтвердить"); // Кнопка подтверждения выбора начальной вершины
    private final JButton setTimeButton = new JButton("Задать интервал");   // Кнопка задания интервала авт. режима
    private final JButton setButton = new JButton("Применить"); // Кнопка установки хар-к графа (веса ребра)
    private final JButton deleteButton = new JButton("Удалить");    // Кнопка удаления вершины/ребра
    private final JToggleButton autoButton = new JToggleButton("Авто"); // Кнопка автоматического режима
    private final JButton stepButton = new JButton("Шаг");  // Кнопка пошагового режима
    private final JButton beginButton = new JButton("Начать");  // Кнопка, запускающая инициализацию алгоритма
    private final JButton resetButton = new JButton("Сброс");   // Кнопка сброса в режим редактирования
    private final JButton clearButton = new JButton("Очистить");    // Кнопка очистки панели с графом
    private final JButton closeButton = new JButton("Закрыть");

    public Window(){
        super();
        init();
    }

    private void init(){
        /*
        Ограничители для менеджера размещения GridBagLayout
        */

        getContentPane().setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();

        /*
        Настройка компонентов 3 уровня (кнопок, текстовых полей и т.п.)
         */

        // Настройка области с выводом данных, добавление возможности её прокрутки
        textArea.setEditable(false);
        textArea.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseEntered(MouseEvent e) {
                super.mouseEntered(e);
                textArea.setCursor(Cursor.getPredefinedCursor(Cursor.TEXT_CURSOR));
            }
        });
        JScrollPane scrollPane = new JScrollPane(textArea,
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        scrollPane.setMinimumSize(new Dimension(-1, 80));
```

```
        // Настройка метки с информацией о действиях
        infoLabel.setHorizontalAlignment(SwingConstants.LEFT);
        infoLabel.setVerticalAlignment(SwingConstants.TOP);
        infoLabel.setBackground(Color.WHITE);
        infoLabel.setOpaque(true);
        infoLabel.setMinimumSize(new Dimension(200, -1));

        // Настройка кнопки загрузки из файла
        loadButton.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseReleased(MouseEvent e) {
                super.mouseReleased(e);
                if(loadButton.isEnabled()) {
                    canvasPanel.load();
                }
            }
        });

        //Настройка кнопки сохранения в файл
        saveButton.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseReleased(MouseEvent e) {
                super.mouseReleased(e);
                if(saveButton.isEnabled()) {
                    canvasPanel.save();
                }
            }
        });

        // Настройка кнопка "Начать"
        beginButton.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseReleased(MouseEvent e) {
                super.mouseReleased(e);
                if(beginButton.isEnabled()) {
                    onUncheck();
                    canvasPanel.setEditable(false);
                    canvasPanel.uncheck();
                    canvasPanel.getParent().repaint();
                    loadButton.setEnabled(false);
                    saveButton.setEnabled(false);
                    beginButton.setEnabled(false);
                    resetButton.setEnabled(true);
                    clearButton.setEnabled(false);
                    textField.setText("");
                    infoLabel.setText("<html>Введите временной интервал<br>(в
миллисекундах)</html>");
                    textField.setVisible(true);
                    setTimeButton.setVisible(true);
                }
            }
        });

        // Настройка кнопки "Сброс"
        resetButton.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
```

```
            super.mouseClicked(e);
            if(resetButton.isEnabled()) {
                solver.reset();
                if (thr != null && thr.isAlive()) {
                    thr.kill();
                }
                canvasPanel.finish();
                canvasPanel.setChoosingInit(false);
                textArea.setText("");
                infoLabel.setText("Информация");
                canvasPanel.getParent().repaint();
                resetButton.setEnabled(false);
                beginButton.setEnabled(true);
                clearButton.setEnabled(true);
                autoButton.setEnabled(false);
                autoButton.setSelected(false);
                stepButton.setEnabled(false);
                saveButton.setEnabled(true);
                loadButton.setEnabled(true);
                textField.setText("");
                textField.setVisible(false);
                setTimeButton.setVisible(false);
                approveButton.setVisible(false);
            }
        }
    });

    // Настройка кнопки "Задать интервал"
    setTimeButton.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseReleased(MouseEvent e) {
            super.mouseReleased(e);
            try {
                int period = Integer.parseInt(textField.getText());
                if(period < 0){
                    infoLabel.setText("<html>Укажите<br>неотрицательное число</html>");
                }
                else {
                    if(thr == null) {
                        thr = new AutoMode(canvasPanel, textArea, autoButton, stepButton, solver,
logger, period);
                    }
                    else if(thr.getState() != Thread.State.NEW){
                        thr.kill();
                        thr = new AutoMode(canvasPanel, textArea, autoButton, stepButton, solver,
logger, period);
                    }
                    textField.setVisible(false);
                    setTimeButton.setVisible(false);
                    infoLabel.setText("Выберите начальную вершину");
                    canvasPanel.setChoosingInit(true);
                    approveButton.setVisible(true);
                }
            }
            catch(NumberFormatException ex){
                infoLabel.setText("<html>Неверный формат,<br>введите заново</html>");
            }
```

```
    }
});

// Настройка кнопки подтверждения выбора начальной вершины
approveButton.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent e) {
        super.mouseReleased(e);
        if (canvasPanel.start()){
            canvasPanel.setChoosingInit(false);
            approveButton.setVisible(false);
            infoLabel.setText("Информация");
            autoButton.setEnabled(true);
            stepButton.setEnabled(true);
        }
        else{
            infoLabel.setText("Вершина не выбрана");
        }
    }
});

// Настройка кнопка "Применить"
setButton.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        super.mouseClicked(e);
        try{
            int weight = Integer.parseInt(textField.getText());
            if(weight < 1){
                infoLabel.setText("<html>Ребро должно иметь<br>положительный вес!</html>");
            }
            else {
                infoLabel.setText("<html>Задать вес ребра /<br>удалить ребро</html>");
                canvasPanel.setEdgeWeight(weight);
            }
        }
        catch (NumberFormatException ex){
            infoLabel.setText("<html>Неверный формат,<br>введите заново</html>");
        }
    }
});

// Настройка конпки "Удалить"
deleteButton.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent e) {
        super.mouseReleased(e);
        canvasPanel.deleteVertex();
        canvasPanel.deleteEdge();
        infoLabel.setText("Информация");
        deleteButton.setVisible(false);
        textField.setVisible(false);
        setButton.setVisible(false);
    }
});

// Настройка кнопки автоматического режима
```

```
autoButton.addItemListener(e -> {
    if(autoButton.isEnabled()) {
        if (e.getStateChange() == ItemEvent.SELECTED) {
            stepButton.setEnabled(false);
            if (thr.isAlive()) {
                thr.enable();
            } else {
                thr.start();
            }
        } else if (e.getStateChange() == ItemEvent.DESELECTED) {
            thr.disable();
            stepButton.setEnabled(true);
        }
    }
});

// Настройка кнопки пошагового режима
stepButton.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        super.mouseClicked(e);
        if (stepButton.isEnabled()) {
            boolean running = solver.step(logger);
            if (!running) {
                if (thr != null && thr.isAlive()) {
                    thr.kill();
                }
                StringBuilder results = new StringBuilder();
                for (String s : solver.results()) {
                    results.append(s).append("\n\n");
                }
                textArea.append("Итоги:\n" + results);
                autoButton.setEnabled(false);
                stepButton.setEnabled(false);
            } else {
                textArea.append(logger.getNextMessage() + "\n");
            }
        }
        canvasPanel.getParent().repaint();
    }
});

// Настройка кнопки "Очистить"
clearButton.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent e) {
        if(clearButton.isEnabled()) {
            super.mouseReleased(e);
            onUncheck();
            onGraphEmpty();
            solver.clear();
            canvasPanel.clear();
        }
    }
});

// Настройка кноки "Закрыть"
```

```
closeButton.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent e) {
        super.mouseReleased(e);
        if(thr != null && thr.isAlive()){
            thr.kill();
        }
        dispose();
    }
});

/*
Настройка компонентов 2 уровня (панелей)
 */

// Помещение прокручиваемой области для вывода данных на панель "Аннотации"
annotationsPanel.setLayout(new GridBagLayout());
annotationsPanel.setBackground(new Color(223,163,159));
annotationsPanel.setBorder(BorderFactory.createTitledBorder("Аннотации"));
gbc.gridx = 0;
gbc.gridy = 0;
gbc.weightx = 1f;
gbc.weighty = 1f;
gbc.fill = GridBagConstraints.BOTH;
gbc.anchor = GridBagConstraints.NORTHWEST;
annotationsPanel.add(scrollPane, gbc);

// Создание верхней панели меню
JMenuBar menuBar = new JMenuBar();
JMenu fileMenu = new JMenu("Файл");

// Настройка кнопки "Закрыть" в меню
JMenuItem closeMenuButton = new JMenuItem("Закрыть");
closeMenuButton.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent e) {
        if(thr != null && thr.isAlive()){
            thr.kill();
        }
        dispose();
        super.mouseReleased(e);
    }
});

fileMenu.add(loadButton);
fileMenu.add(saveButton);
fileMenu.add(closeMenuButton);
menuBar.add(fileMenu);

// Настройка панели отображения графа
canvasPanel.setMinimumSize(new Dimension(-1, 100));
canvasPanel.setBorder(BorderFactory.createTitledBorder("Граф"));
canvasPanel.setBackground(new Color(151,248,255));
canvasPanel.setOpaque(true);

// Добавление компонентов на разделённую панель
splitPane.setTopComponent(annotationsPanel);
```

```
splitPane.setBottomComponent(canvasPanel);
splitPane.setDividerLocation(105);

// Настройка панели установок
settingsPanel.setLayout(new GridBagLayout());
settingsPanel.setBackground(new Color(222,227,119));
settingsPanel.setBorder(BorderFactory.createTitledBorder("Установки"));
settingsPanel.setPreferredSize(new Dimension(200, -1));

// Добавление элементов на панель установок
gbc.gridx = 0;
gbc.gridy = 0;
gbc.gridwidth = 2;
gbc.weightx = 1f;
gbc.weighty = 0.005;
gbc.anchor = GridBagConstraints.NORTH;
gbc.fill = GridBagConstraints.BOTH;
settingsPanel.add(infoLabel, gbc);
gbc.gridy = 1;
settingsPanel.add(textField, gbc);
gbc.gridy = 2;
settingsPanel.add(approveButton, gbc);
gbc.gridwidth = 1;
gbc.weightx = 0.4f;
settingsPanel.add(setButton, gbc);
gbc.gridx = 1;
settingsPanel.add(deleteButton, gbc);
gbc.gridx = 0;
gbc.gridwidth = 2;
gbc.weightx = 1f;
settingsPanel.add(setTimeButton, gbc);
gbc.gridy = 4;
gbc.gridwidth = 1;
gbc.weighty = 0.495;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.anchor = GridBagConstraints.SOUTH;
gbc.weightx = 0.4f;
settingsPanel.add(autoButton, gbc);
gbc.gridx = 1;
settingsPanel.add(stepButton, gbc);

// Настройка нижней панели
bottomPanel.setLayout(new GridBagLayout());
bottomPanel.setBorder(BorderFactory.createEmptyBorder(3,3,3,3));
bottomPanel.setBackground(new Color(140, 223, 122));

// Добавление элементов на нижнюю панель
gbc.anchor = GridBagConstraints.CENTER;
gbc.gridx = 0;
gbc.gridy = 0;
gbc.weightx = 0.02f;
gbc.weighty = 0f;
bottomPanel.add(beginButton, gbc);
gbc.gridx = 1;
bottomPanel.add(resetButton, gbc);
gbc.gridx = 2;
gbc.weightx = 1f;
```

```
bottomPanel.add(Box.createHorizontalStrut(0), gbc);
gbc.gridx = 3;
gbc.weightx = 0.02f;
bottomPanel.add(clearButton, gbc);
gbc.gridx = 4;
gbc.weightx = 0.02f;
bottomPanel.add(closeButton, gbc);

/*
Настройка компонентов 1 уровня (главная панель)
 */

getContentPane().setBackground(Color.GRAY);

// Добавление меню на окно
gbc.gridx = 0;
gbc.gridy = 0;
gbc.weightx = 1f;
gbc.weighty = 0f;
gbc.anchor = GridBagConstraints.NORTHWEST;
gbc.gridwidth = GridBagConstraints.REMAINDER;
gbc.fill = GridBagConstraints.HORIZONTAL;
getContentPane().add(menuBar, gbc);

// Добавление разделённой панели (Аннотации/Граф) на окно
gbc.gridx = 0;
gbc.gridy = 1;
gbc.weightx = 1f;
gbc.weighty = 1f;
gbc.gridwidth = 1;
gbc.fill = GridBagConstraints.BOTH;
gbc.insets.set(3,3,3,0);
getContentPane().add(splitPane, gbc);

// Добавление панели установок на окно
gbc.gridx = 1;
gbc.weightx = 0f;
gbc.gridheight = 2;
gbc.insets.set(3,3,3,3);
getContentPane().add(settingsPanel, gbc);

// Добавление нижней панели на окно
gbc.gridx = 0;
gbc.gridy = 2;
gbc.weightx = 1f;
gbc.weighty = 0f;
gbc.gridheight = 1;
gbc.insets.set(0,3,3,0);
getContentPane().add(bottomPanel, gbc);

/*
Настройка компонента 0 уровня (окна)
 */

setTitle("Алгоритм Дейкстры");
setMinimumSize(new Dimension(700,700));
setLocationRelativeTo(null);
```

```
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    pack();

    saveButton.setEnabled(false);
    beginButton.setEnabled(false);
    resetButton.setEnabled(false);
    clearButton.setEnabled(false);
    textField.setVisible(false);
    approveButton.setVisible(false);
    setTimeButton.setVisible(false);
    setButton.setVisible(false);
    deleteButton.setVisible(false);
    autoButton.setEnabled(false);
    stepButton.setEnabled(false);

    setVisible(true);
}

// При выборе вершины
public void onVertexChoice(int id){
    infoLabel.setText("Выбрана вершина " + id);
    deleteButton.setVisible(true);
}

// При выборе ребра
public void onEdgeChoice(int weight){
    infoLabel.setText("<html>Задать вес ребра /<br>удалить ребро</html>");
    textField.setText(Integer.toString(weight));
    textField.setVisible(true);
    deleteButton.setVisible(true);
    setButton.setVisible(true);
}

// При снятии выбора
public void onUncheck(){
    infoLabel.setText("Информация");
    textField.setVisible(false);
    setButton.setVisible(false);
    deleteButton.setVisible(false);
}

// Если граф содержит хотя бы одну вершину
public void onGraphNotEmpty(){
    saveButton.setEnabled(true);
    beginButton.setEnabled(true);
    clearButton.setEnabled(true);
}

// Если из графа удалены все вершины
public void onGraphEmpty(){
    saveButton.setEnabled(false);
    beginButton.setEnabled(false);
    clearButton.setEnabled(false);
}

public static void main(String[] args){
    new Window();
```

```
    }
}
```

GPanel.java:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.geom.AffineTransform;
import java.util.ArrayList;
import java.util.Comparator;

public class GPanel extends JPanel {

    private final Font FONT = new Font("TimesRoman", Font.BOLD, 14); // Шрифт надписей на
вершинах
    private final int RADIUS = 20; // Радиус вершин
    private final Solver solver;    // Объект, выполняющий алгоритм
    private final ArrayList<VisualVertex> vertices = new ArrayList<>(); // Список вершин
    private final ArrayList<VisualEdge> edges = new ArrayList<>();  // Список рёбер
    private VisualVertex chosenVertex = null;   // Выбранная вершина
    private VisualVertex initVertex = null;     // Начальная вершина
    private VisualVertex lastConsideredVertex = null;   // Последняя рассмотренная вершина
    private VisualVertex draggedVertex = null;  // Перетаскиваемая вершина
    private VisualEdge chosenEdge = null;   // Выбранное ребро
    private VisualEdge drawnEdge = null;    // Рисуемое ребро
    boolean drawingEdge = false;    // Режим рисования ребра
    boolean holdingVertex = false;  // Вершина зажата/отпущена
    boolean draggingVertex = false; // Режим перетаскивания вершин
    private boolean isEditable = true;  // Режим редактирования графа
    private boolean choosingInit = false;   // Режим выбора начальной вершины

    public GPanel(Solver sol){
        super();

        this.solver = sol;

        addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {
                super.keyPressed(e);
                if(isEditable) {
                    if (e.getKeyCode() == KeyEvent.VK_SHIFT) {  // Включение режима рисования
ребра
                        drawingEdge = true;
                    }
                }
            }

            @Override
            public void keyReleased(KeyEvent e) {
                super.keyReleased(e);
                if(isEditable) {
                    if (e.getKeyCode() == KeyEvent.VK_SHIFT) {  // Выключение режима рисования
ребра
                        drawingEdge = false;
                    }
                }
```

```
        }
    });

    addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            super.mouseClicked(e);
            if(isEditable) {
                Window window = (Window) getTopLevelAncestor();
                if ((e.getClickCount() == 2) && (e.getButton() == 1)) { // Добавление вершины
                    boolean intersects = false;
                    for (VisualVertex v : vertices) {
                        if ((Math.pow((e.getX() - v.getX()), 2) + Math.pow((e.getY() - v.getY()), 2)) <
                                Math.pow(2 * (double) RADIUS, 2) + 10) {
                            intersects = true;
                            break;
                        }
                    }
                    if (!intersects && !(e.getX() < RADIUS + 10 || e.getY() < RADIUS + 10
                            || e.getX() > getSize().width - RADIUS - 10 || e.getY() > getSize().height -
RADIUS - 10)) {
                        vertices.add(new VisualVertex(e.getX(), e.getY(), solver.addVertex()));
                        if (vertices.size() == 1) {
                            window.onGraphNotEmpty();
                        }
                    }
                } else if ((e.getClickCount() == 1) && (e.getButton() == 1)) {  // Выбор вершины или
ребра
                    window.onUncheck();
                    chosenVertex = chooseCircle(e.getX(), e.getY());
                    if (chosenVertex == null) {
                        ArrayList<VisualEdge> toChose = new ArrayList<>(5);
                        for (VisualEdge edge : edges) {
                            if (edge.getLine().contains(e.getX(), e.getY())) {
                                toChose.add(edge);
                            }
                        }
                        double closeEnough = 0.0;
                        if (toChose.isEmpty()) {
                            chosenEdge = null;
                        }
                        for (VisualEdge edge : toChose) {
                            double curDistToPoint = Math.sqrt(Math.pow(e.getX() - edge.getV1().getX(), 2)
+ Math.pow(e.getY() - edge.getV1().getY(), 2))
                                    + Math.sqrt(Math.pow(e.getX() - edge.getV2().getX(), 2) +
Math.pow(e.getY() - edge.getV2().getY(), 2));
                            double diff = curDistToPoint - Math.sqrt(Math.pow(edge.getV2().getX() -
edge.getV1().getX(), 2) + Math.pow(edge.getV2().getY() - edge.getV1().getY(), 2));
                            if (closeEnough == 0.0) {
                                closeEnough = diff;
                                chosenEdge = edge;
                            } else if (diff < closeEnough) {
                                closeEnough = diff;
                                chosenEdge = edge;
                            }
                        }
                    } else {
```

```
                    chosenEdge = null;
                    window.onVertexChoice(chosenVertex.getId());
                }
                if (chosenEdge != null) {
                    window.onEdgeChoice(chosenEdge.getWeight());
                }
            }
            getParent().repaint();
        }
        else if(choosingInit){
            initVertex = chooseCircle(e.getX(), e.getY());
            getParent().repaint();
        }
    }

    @Override
    public void mousePressed(MouseEvent e) {
        super.mousePressed(e);
        if(isEditable) {
            if (drawingEdge) {  // Начать рисование ребра
                VisualVertex vertex = new VisualVertex(e.getX(), e.getY(), 0);
                drawnEdge = new VisualEdge(vertex, vertex);
                getParent().repaint();
            }
            draggedVertex = chooseCircle(e.getX(), e.getY());
            if (draggedVertex != null)  // Обнаружить зажатую вершину
                holdingVertex = true;
        }
    }

    @Override
    public void mouseReleased(MouseEvent e) {
        super.mouseReleased(e);
        if(isEditable) {
            holdingVertex = false;  // Изначально зажатая вершина больше не берётся во
внимание
            if (drawingEdge && drawnEdge != null && (e.getX() != drawnEdge.getV1().getX() ||
e.getY() != drawnEdge.getV1().getY())) {   // Рисование ребра
                boolean isInFirst = false;
                boolean isInSecond = false;
                VisualVertex first = null;
                VisualVertex second = null;
                for (VisualVertex v : vertices) {
                    if ((Math.pow((drawnEdge.getV1().getX() - v.getX()), 2) +
Math.pow((drawnEdge.getV1().getY() - v.getY()), 2)) < RADIUS * RADIUS) {
                        first = v;
                        isInFirst = true;
                    }
                    if ((Math.pow((drawnEdge.getV2().getX() - v.getX()), 2) +
Math.pow((drawnEdge.getV2().getY() - v.getY()), 2)) < RADIUS * RADIUS) {
                        second = v;
                        isInSecond = true;
                    }
                }
                if (isInFirst && isInSecond && first != second) {
                    if (first.getId() < second.getId()) {
                        drawnEdge.setV1(first);
```

```
                    drawnEdge.setV2(second);
                } else {
                    drawnEdge.setV2(first);
                    drawnEdge.setV1(second);
                }
                boolean exists = false;
                for (VisualEdge edge : edges) {
                    if (drawnEdge.equals(edge)) {
                        exists = true;
                        break;
                    }
                }
                if (!exists) {
                    setShape(drawnEdge);
                    edges.add(drawnEdge);
                    solver.addEdge(drawnEdge.getV1().getId(), drawnEdge.getV2().getId(),
drawnEdge.getWeight());
                }
            }
        }
        drawnEdge = null;
        getParent().repaint();
    }
}

@Override
public void mouseEntered(MouseEvent e){
    super.mouseEntered(e);
    if(isEditable) {
        setFocusable(true);
        requestFocusInWindow();
    }
}

public void mouseExited(MouseEvent e){
    super.mouseExited(e);
    if(isEditable) {
        drawingEdge = false;
        setFocusable(false);
    }
}
});

addMouseMotionListener(new MouseAdapter() {
    @Override
    public void mouseDragged(MouseEvent e) {
        super.mouseDragged(e);
        if(isEditable) {
            if (drawingEdge && drawnEdge != null) {
                drawnEdge.setV2(new VisualVertex(e.getX(), e.getY(), 0));   // Обновление
рисуемого ребра
            } else
                drawnEdge = null;

            if (holdingVertex && !drawingEdge) {    // Перетаскивание вершины
                draggingVertex = false;
                for (VisualEdge edge : edges) {
```

```
                    if (edge.getV1().getId() == draggedVertex.getId()) {
                        draggedVertex.setX(e.getX());
                        draggedVertex.setY(e.getY());
                        edge.setV1(draggedVertex);
                        setShape(edge);
                        draggingVertex = true;
                    }
                    if (edge.getV2().getId() == draggedVertex.getId()) {
                        draggedVertex.setX(e.getX());
                        draggedVertex.setY(e.getY());
                        edge.setV2(draggedVertex);
                        setShape(edge);
                        draggingVertex = true;
                    }
                }
                if (!draggingVertex) {
                    draggedVertex.setX(e.getX());
                    draggedVertex.setY(e.getY());
                }
            }
            getParent().repaint();
        }
    }
    });
}

public void clear(){ // Удаление графа
    vertices.clear();
    edges.clear();
    chosenVertex = null;
    chosenEdge = null;
    initVertex = null;
    lastConsideredVertex = null;
    solver.clear();
    getParent().repaint();
}

public void uncheck(){  // Снять выбор со вершины или ребра
    this.chosenVertex = null;
    this.chosenEdge = null;
}

private VisualVertex chooseCircle(int x, int y) // Найти вершину на данной позиции
{
    for (VisualVertex vertex : vertices) {
        if ((Math.pow((x - vertex.getX()), 2) + Math.pow((y - vertex.getY()), 2)) < RADIUS*RADIUS
+ 1) {
            return vertex;
        }
    }
    return null;
}

public void setChoosingInit(boolean flag){  // Режим выбора начальной вершины
    choosingInit = flag;
}
```

```
    public void setEditable(boolean flag){  // Режим редактирования графа
        isEditable = flag;
    }

    public boolean start(){ // Начало работы алгоритма
        if(initVertex == null){
            return false;
        }
        solver.setInit(initVertex.getId());
        getParent().repaint();
        return true;
    }

    public void finish(){   // Завершение работы алгоритма
        initVertex = null;
        isEditable = true;
        getParent().repaint();
    }

    public void deleteVertex(){ // Удаление выбранной вершины
        if(chosenVertex != null) {
            int index = vertices.indexOf(chosenVertex);
            vertices.remove(chosenVertex);
            edges.removeIf(edge -> chosenVertex.getId() == edge.getV1().getId() ||
chosenVertex.getId() == edge.getV2().getId());
            for(int i = index; i < vertices.size(); i++){
                vertices.get(i).setId(i + 1);
            }
            solver.deleteVertex(chosenVertex.getId());
            chosenVertex = null;
            if(vertices.isEmpty()){
                Window window = (Window)getTopLevelAncestor();
                window.onGraphEmpty();
            }
            getParent().repaint();
        }
    }

    public void deleteEdge(){   // Удаление выбранного ребра
        if(chosenEdge != null){
            edges.remove(chosenEdge);
            solver.deleteEdge(chosenEdge.getV1().getId(), chosenEdge.getV2().getId());
            chosenEdge = null;
            getParent().repaint();
        }
    }

    private void setShape(VisualEdge edgeDrawn){    // Установка фигуры для отображения
ребра
        int newWidth = (int)Math.sqrt((Math.pow(edgeDrawn.getV2().getX() -
edgeDrawn.getV1().getX(),2))
                + (Math.pow(edgeDrawn.getV2().getY() - edgeDrawn.getV1().getY(),2)));
        int newX = (edgeDrawn.getV2().getX() + edgeDrawn.getV1().getX())/2 - newWidth/2;
        int newY = (edgeDrawn.getV2().getY() + edgeDrawn.getV1().getY())/2 - 1;
        int dX = edgeDrawn.getV2().getX() - edgeDrawn.getV1().getX();
        int dY = edgeDrawn.getV2().getY() - edgeDrawn.getV1().getY();
        Rectangle rect = new Rectangle(newX, newY, newWidth, 3);
```

```
        AffineTransform at = new AffineTransform();
        at.rotate(Math.atan((double)dY/dX), newX + (float)newWidth/2, newY + 1.5f);
        Shape shape = at.createTransformedShape(rect);

        edgeDrawn.setLine(shape);
    }

    public void load(){ // Загрузка из файла
        FileHandler fh = new FileHandler();
        ArrayList<Integer> loaded =  fh.load();
        if(!loaded.isEmpty()) {
            clear();
            int number = loaded.get(0);
            int vId = 1;
            for (int i = 0; i < number; i++) {
                vertices.add(new VisualVertex(loaded.get(vId + i*2), loaded.get(vId + i*2 + 1),
solver.addVertex()));
            }
            for (int j = number*2 + 1; j < loaded.size(); j++) {
                if (loaded.get(j) == -1) {
                    vId++;
                } else {
                    int destVer = loaded.get(j++);
                    int weight = loaded.get(j);
                    solver.addEdge(vId, destVer, weight);
                    VisualEdge addedEdge = new VisualEdge(vertices.get(vId - 1), vertices.get(destVer -
1));
                    addedEdge.setWeight(weight);
                    setShape(addedEdge);
                    edges.add(addedEdge);
                }
            }
            Window window = (Window)getTopLevelAncestor();
            window.onGraphNotEmpty();
        }
    }

    public void save(){ // Сохранение в файл
        edges.sort(edgeComparator);
        FileHandler fh = new FileHandler();
        fh.save(vertices, edges);
    }

    // Компаратор для сортировки рёбер
    private static final Comparator<VisualEdge> edgeComparator = Comparator.comparingInt(o ->
o.getV1().getId());

    public void setEdgeWeight(int w){   // Установка веса выбранного ребра
        chosenEdge.setWeight(w);
        solver.setEdgeWeight(chosenEdge.getV1().getId(), chosenEdge.getV2().getId(), w);
        getParent().repaint();
    }

    public void paintComponent(Graphics g2){    // Отрисовка графа
        Graphics2D g = (Graphics2D)g2;
        super.paintComponent(g);
```

```
        // Отрисовка рёбер графа с учётом текущего пути
        lastConsideredVertex = null;
        for(VisualVertex v : vertices){
            if(solver.getVertex(v.getId()).getColor() == Colors.COLOR2){
                lastConsideredVertex = v;
                break;
            }
        }
        ArrayList<Integer> path = new ArrayList<>();
        if(lastConsideredVertex != null){
            path.add(lastConsideredVertex.getId());
            Vertex par = solver.getVertex(lastConsideredVertex.getId()).getParent();
            while(par != null){
                path.add(0, par.getId());
                par = par.getParent();
            }
        }
        for(VisualEdge e : edges){
            g.setColor(Color.BLACK);
            if(path.contains(e.getV1().getId())){
                int ind = path.indexOf(e.getV1().getId());
                if((ind < path.size() - 1 && path.get(ind + 1) == e.getV2().getId()) || (ind > 0 &&
path.get(ind - 1) == e.getV2().getId())){
                    g.setColor(Color.RED);
                }
            }
            g.draw(e.getLine());
            g.fill(e.getLine());
            g.setColor(Color.BLACK);
            String weight = Integer.toString(e.getWeight());
            double tan = (double)(e.getV2().getY() - e.getV1().getY())/(e.getV2().getX() -
e.getV1().getX());
            int xOffset = (e.getV1().getX() + e.getV2().getX())/2 - weight.length()*6 - 11;
            int yOffset;
            if(tan < 0){
                yOffset = (e.getV1().getY() + e.getV2().getY())/2 - 6;
            }
            else{
                yOffset = (e.getV1().getY() + e.getV2().getY())/2 + 14;// + weight.length()*10;
            }

            g.drawString(weight, xOffset, yOffset);
        }

        // Отрисовка выбранного ребра
        if(chosenEdge != null){
            g.setColor(Color.ORANGE);
            g.draw(chosenEdge.getLine());
            g.fill(chosenEdge.getLine());
        }

        // Отрисовка всех вершин
        for(VisualVertex p: vertices) {
            Color col = Color.BLACK;
            Colors color = solver.getVertex(p.getId()).getColor();
            int pathLen = solver.getVertex(p.getId()).getPathLen();
            String pLen;
```

```
            if(pathLen == Integer.MAX_VALUE){
                pLen = "\u221E";
            }
            else{
                pLen = Integer.toString(pathLen);
            }
            switch (color){
                case COLOR1 -> col = Color.WHITE;//Было GREEN
                case COLOR2 -> {
                    col = Color.PINK;
                    lastConsideredVertex = p;
                }
                case COLOR3 -> col = Color.ORANGE;
                case COLOR4 -> col = Color.lightGray;//Было BLUE
            }
            g.setColor(col);
            g.fillOval(p.getX() - RADIUS, p.getY() - RADIUS, RADIUS*2, RADIUS*2);
            g.setColor(Color.BLACK);
            g.drawOval(p.getX() - RADIUS, p.getY() - RADIUS, RADIUS*2, RADIUS*2);
            String s = Integer.toString(p.getId());
            g.setFont(FONT);
            g.drawString(s, p.getX() - 3 - 4*(s.length() - 1), p.getY() + 6);
            g.setColor(Color.WHITE);
            g.fillRect(p.getX() - 37 - 7*(pLen.length() - 1), p.getY() + 6, 8*(pLen.length() + 1), 14);
            g.setColor(Color.BLACK);
            g.drawRect(p.getX() - 37 - 7*(pLen.length() - 1), p.getY() + 6, 8*(pLen.length() + 1), 14);
            g.drawString(pLen, p.getX() - 35 - 7*(pLen.length() - 1), p.getY() + 18);
        }

        // Отрисовка рисуемого ребра
        if(drawnEdge != null){
            g.drawLine(drawnEdge.getV1().getX(), drawnEdge.getV1().getY(),
drawnEdge.getV2().getX(), drawnEdge.getV2().getY());
        }

        // Отрисовка начальной вершины
        if(initVertex != null){
            g.setColor(Color.RED);
            g.setStroke(new BasicStroke(3));
            g.drawOval(initVertex.getX() - RADIUS, initVertex.getY() - RADIUS, RADIUS*2,
RADIUS*2);
        }

        // Отрисовка выбранной вершины
        if(chosenVertex != null){
            g.setColor(Color.YELLOW);
            g.setStroke(new BasicStroke(3));
            g.drawOval(chosenVertex.getX() - RADIUS, chosenVertex.getY() - RADIUS, RADIUS*2,
RADIUS*2);
        }
    }
}
```

Solver.java:

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.Map;
import java.util.PriorityQueue;

// Исполнитель алгоритма
public class Solver {
    private Vertex init;    // Начальная вершина
    private Vertex prev;    // Предыдущая рассмотренная вершина
    private final ArrayList<Vertex> vertSet;    // Вершины графа
    private final PriorityQueue<Vertex> front;  // Досягаемые на данный момент вершины

    public Solver() {
        this.init = null;
        this.prev = null;
        this.vertSet = new ArrayList<>();
        this.front = new PriorityQueue<>(1, distComp);
    }

    // Компаратор для вершин, определяет, какая из досягаемых выбирается на текущем шаге
    private final static Comparator<Vertex> distComp = (o1, o2) -> {
        if(o1.getPathLen() == o2.getPathLen()){
            return o1.getId() - o2.getId();
        }
        return o1.getPathLen() - o2.getPathLen();
    };

    // Добавить вершину
    public int addVertex() {
        this.vertSet.add(new Vertex(vertSet.size() + 1));
        return this.vertSet.size();
    }

    // Удалить вершину с указанным ID
    public void deleteVertex(int id) {
        Vertex toDel = this.vertSet.get(id - 1);
        for (Map.Entry<Vertex, Integer> pair : toDel.getAdjList().entrySet()) {
            pair.getKey().getAdjList().remove(toDel);
        }
        this.vertSet.remove(toDel);
        for (int i = id; i < this.vertSet.size() + 1; i++) {
            this.vertSet.get(i - 1).setId(i);
        }
    }

    // Получить вершину с указанным ID
    public Vertex getVertex(int id) {
        return this.vertSet.get(id - 1);
    }

    // Добавить ребро
    public void addEdge(int from, int to, int dist){
        Vertex fromVer = this.vertSet.get(from - 1);
        Vertex toVer = this.vertSet.get(to - 1);
        if(!fromVer.getAdjList().containsKey(toVer)) {
            fromVer.addToAdjList(toVer, dist);
```

```
            toVer.addToAdjList(fromVer, dist);
        }
    }

    // Установить вес указанного ребра
    public void setEdgeWeight(int from, int to, int weight){
        Vertex s = getVertex(from);
        Vertex d = getVertex(to);
        if(d != null && s != null && s.getAdjList().containsKey(d)) {
            s.getAdjList().replace(d, weight);
            d.getAdjList().replace(s, weight);
        }
    }

    // Удалить указанное ребро
    public void deleteEdge(int from, int to){
        Vertex fromVer = this.vertSet.get(from - 1);
        Vertex toVer = this.vertSet.get(to - 1);
        fromVer.getAdjList().remove(toVer);
        toVer.getAdjList().remove(fromVer);
    }

    // Назначить начальную вершину
    public void setInit(int init){
        this.init = this.vertSet.get(init - 1);
        this.front.add(this.init);
    }

    // Выполнение одного шага алгоритма
    public boolean step(CustomLogger logger){
        if(prev != null){
            prev.setColor(Colors.COLOR4);
        }
        if(!(front.isEmpty())){
            Vertex current = front.poll();
            current.setColor(Colors.COLOR2);
            prev = current;
            if(current == init){
                current.setPathLen(0);
            }
            logger.addMessage(formInfo(current));

            // Проверка на начличие путей из начальной вершины
            if (current.getAdjList().isEmpty()) {
                logger.addMessage("Начальная вершина не имеет смежных\n");
            }
            else {
                logger.addMessage("Рассматриваются смежные вершины: \n");
            }
            for(Map.Entry<Vertex, Integer> next : current.getAdjList().entrySet()) {

                // Проверка на факт рассмотрения вершины ранее
                if (next.getKey().getColor() == Colors.COLOR4) {
                    logger.addMessage("--Вершина " + next.getKey().getId() + " уже рассмотрена!\n");
                }
                // Проверка на изменение текущей длины пути до данной вершины
                if ((next.getKey().getColor() != Colors.COLOR4) &&
```

```
                    (current.getPathLen() + next.getValue() < next.getKey().getPathLen())) {
                    logger.addMessage("--Длина пути для вершины " + next.getKey().getId() + "
меняется с " +

(next.getKey().getPathLen()==Integer.MAX_VALUE?"\u221E":next.getKey().getPathLen())
                            + " на " + (current.getPathLen() + next.getValue()) + "\n");
                next.getKey().setParent(current);
                next.getKey().setPathLen(current.getPathLen() + next.getValue());
                if (!(front.contains(next.getKey()))) {
                    next.getKey().setColor(Colors.COLOR3);
                    front.add(next.getKey());
                }
            }
            else if ((next.getKey().getColor() != Colors.COLOR4) &&
                    (current.getPathLen() + next.getValue() >= next.getKey().getPathLen())) {
                logger.addMessage("--Для вершины " + next.getKey().getId() + " длина пути
остается прежней\n");
            }
        }
        return true;
    }
    return false;
}

// Очистка графа
public void clear(){
    this.init = null;
    this.prev = null;
    vertSet.clear();
    front.clear();
}

// Сброс графа до начального состояния
public void reset(){
    this.init = null;
    this.prev = null;
    for(Vertex v : vertSet){
        v.setPathLen(Integer.MAX_VALUE);
        v.setParent(null);
        v.setColor(Colors.COLOR1);
    }
    front.clear();
}

// Форматирование информации о вершине
private String formInfo(Vertex v){
    String info = v.getId() + ". ";
    StringBuilder path = new StringBuilder();
    Vertex par = v.getParent();
    if(par == null && !(v == init)){
        path = new StringBuilder("Путь: не существует\n");
        info = info + path + "Длина пути: " + "\u221E";
    }
    else {
        while (!(par == null)) {
            path.insert(0, par.getId() + " ");
            par = par.getParent();
```

```
            }
            path = new StringBuilder("Путь: " + path + v.getId() + "\n");
            info = info + path + "Длина пути: " + v.getPathLen() + "\n";
        }
        return info;
    }

    // Получение конечных результатов
    public ArrayList<String> results(){
        ArrayList<String> res = new ArrayList<>(0);
        for(Vertex v : this.vertSet){
            res.add(formInfo(v));
        }
        return res;
    }
}
```

CustomLogger.java:

```
import java.util.ArrayList;

// Объект, предоставляющий записанные данные
public class CustomLogger {
    private final ArrayList<String> messages;
    private int nextMessageIndex;
    private boolean endReached;

    public CustomLogger(){
        this.messages = new ArrayList<>();
        this.nextMessageIndex = 0;
        this.endReached = true;
    }

    // Добавить строку с сообщением
    public void addMessage(String message){
        if(this.isEndReached()){
            this.endReached = false;
        }
        this.messages.add(message);
    }

    // Получить следующее сообщение
    public String getNextMessage(){
        if(this.isEndReached()){
            return "";
        }
        StringBuilder ret = new StringBuilder();
        for (;nextMessageIndex < (this.messages.size() - 1); nextMessageIndex++){
            ret.append( this.messages.get(this.nextMessageIndex));
        }
        this.endReached = true;
        ret.append( this.messages.get(this.nextMessageIndex++));

        return ret.toString();
    }

    // Достигнут ли конец списка сообщений
    public boolean isEndReached(){
        return this.endReached;
    }
}
```

FileHandler.java:

```
import javax.swing.*;
import java.io.*;
import java.util.ArrayList;
import java.util.Scanner;

public class FileHandler {
    private File file;

    public FileHandler(){
        file = null;
    }

    public void save(ArrayList<VisualVertex> vertices, ArrayList<VisualEdge> edges) { //
Сохранение в файл
        JFileChooser chooser = new JFileChooser();
        chooser.showDialog(null, "Сохранить");
        file = chooser.getSelectedFile();
        try (FileWriter fw = new FileWriter(file)) {
            fw.write(vertices.size() + "\n");
            for(VisualVertex v : vertices){
                fw.write(v.getX() + " " + v.getY() + "\n");
            }
            int vId = 1;
            for(VisualEdge e : edges){
                while(e.getV1().getId() != vId){
                    vId++;
                    fw.write("/\n");
                }
                fw.write(e.getV2().getId() + " " + e.getWeight() + "\n");
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public ArrayList<Integer> load(){   // Загрузка из файла
        JFileChooser chooser = new JFileChooser();
        chooser.showDialog(null, "Загрузить");
        file = chooser.getSelectedFile();
        ArrayList<Integer> input = new ArrayList<>();
        try{
            Scanner scanner = new Scanner(file);
            String next = scanner.next();
            while(scanner.hasNext()){
                if(next.equals("/")){
                  input.add(-1);
                }
                else{
                    input.add(Integer.parseInt(next));
                }
                next = scanner.next();
            }
            if(next.equals("/")){
                input.add(-1);
            }
            else{
```

```
                input.add(Integer.parseInt(next));
            }
        }
        catch(FileNotFoundException ex){
            ex.printStackTrace();
        }
        return input;
    }
}
```

Vertex.java:

```
import java.util.HashMap;

enum Colors{COLOR1, COLOR2, COLOR3, COLOR4}

// Вершина графа для исполнителя
public class Vertex {
    private int id;
    private int pathLen;
    private Vertex parent;
    private final HashMap<Vertex, Integer> adjList;
    private Colors color;

    public Vertex(int id){
        this.id = id;
        this.pathLen = Integer.MAX_VALUE;
        this.parent = null;
        this.adjList = new HashMap<>();
        this.color = Colors.COLOR1;
    }

    // Добавить другую вершину в список смежности
    public void addToAdjList(Vertex v, Integer dist){
        this.adjList.put(v, dist);
    }

    public void setPathLen(int pathLen){
        this.pathLen = pathLen;
    }

    public void setParent(Vertex v){
        this.parent = v;
    }

    public void setColor(Colors color){
        this.color = color;
    }

    public void setId(int id){
        this.id = id;
    }

    public int getId(){
        return this.id;
    }

    public int getPathLen(){
        return this.pathLen;
    }

    public Vertex getParent(){
        return this.parent;
    }

    public Colors getColor(){
        return this.color;
    }
```

```
    public HashMap<Vertex, Integer> getAdjList(){
        return this.adjList;
    }
}
```

VisualEdge.java:

```
import java.awt.*;

// Ребро для отрисовки
public class VisualEdge {
    private VisualVertex v1;
    private VisualVertex v2;
    private Shape line = null;
    private int weight = 1;

    public VisualEdge(VisualVertex v1, VisualVertex v2){
        this.v1 = v1;
        this.v2 = v2;
    }

    public boolean equals(VisualEdge other){
        return (v1 == other.getV1() && v2 == other.getV2());
    }

    public void setV1(VisualVertex v1){
        this.v1 = v1;
    }

    public void setV2(VisualVertex v2){
        this.v2 = v2;
    }

    public void setLine(Shape shape){
        this.line = shape;
    }

    public void setWeight(int w){
        this.weight = w;
    }

    public VisualVertex getV1(){
        return this.v1;
    }

    public  VisualVertex getV2(){
        return this.v2;
    }

    public Shape getLine(){
        return this.line;
    }

    public int getWeight(){
        return this.weight;
    }
}
```

VisualVertex.java:

```
// Вершина для отрисовки
public class VisualVertex {
    private int x;
    private int y;
    private int id;

    public VisualVertex(int x, int y, int id){
        this.x = x;
        this.y = y;
        this.id = id;
    }

    public void setX(int x){
        this.x = x;
    }

    public void setY(int y){
        this.y = y;
    }

    public void setId(int id){
        this.id = id;
    }

    public int getX(){
        return this.x;
    }

    public int getY(){
        return this.y;
    }

    public int getId(){
        return this.id;
    }
}
```