Санкт-Петербургский политехнический университет

Институт компьютерных наук и технологий

**Кафедра «Компьютерные системы и программные технологии»**

**КУРСОВОЙ ПРОЕКТ**

**Дисциплина:** Алгоритмы и структуры данных

Тема: Разработка GUI приложения для нахождения маршрута, который проходит по каждому ребру графа один раз

Выполнил студент гр. 3530901/90003 Щигорева Д.С.

Руководитель Ахин М.Х.

Санкт-Петербург

2019

### Оглавление

|  |  |
| --- | --- |
| ТЕХНИЧЕСКОЕ ЗАДАНИЕ................................................................................................................ | 4 |
| МЕТОД РЕШЕНИЯ…………………………….................................................... | 5 |
| СКРИНШОТЫ ПРОГРАММЫ …….................................................................... | 6 |
| ЛИТЕРАТУРА ….................................................................................................... | 7 |

### ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Задан неориентированный граф. Найти маршрут, если такой есть, который проходит по каждому ребру графа один раз. Вывести найденный маршрут.

GitHub репозиторий проекта: <https://github.com/schigoreva/algo-euler>

### МЕТОД РЕШЕНИЯ

В проекте использована концепция MVC (Model-View-Controller) для отделения бизнес-логики от визуализации. Код разделен на Main.java Preferences.java и три пакета: view, model, test.

Main.java – запуск приложения.

Preferences.java – класс для констант, используемые во всем проекте.

Содержимое пакета view:

* ControlView.java – отвечает за вызов главного окна, представление графа на нем и найденного маршрута.
* EdgeView.java – отвечает за изображение ребра.
* GraphView.java -
* InfoView.java – отвечает за окно с информацией о программе, которое открывается при нажатии на кнопку «Информация».
* VertexView.java – отвечает за изображения вершины, при создании графа.
* ViewPreferences.java – класс для констант, используемые в пакете view.

Содержимое пакета model:

* GraphModel.java – отвечет за создание и удаление ребер и вершин, а также за алгоритм нахождения маршрута.

Содержимое пакета test:

* Tests.java – отвечает за тестирование функций, используемых в программе.

Более подробная работа программы:

В главном окне пользователь с помощью панели справа и главного поля, задает неориентированный граф. На главном поле можно расставить вершины, а благодаря панели справа можно создать ребра между вершинами, записав в пустые поля номера вершин и нажав на кнопку «Создать ребро».

В случае, если в поле ввести некорректные входные данные, пользователю вылетит сообщение об этом, в правом нижнем углу.

После создания графа, для получения маршрута, пользователь должен нажать кнопку «Найти маршрут». Если в заданном графе невозможно найти маршрут, который проходит по каждому ребру графа один раз, пользователь получит сообщение об этом.

При создании вершин или ребер, пользователь также получает сообщение об успешной выполненной задаче.

При нажатии на кнопку «Сброс» граф удаляется и главное поле становится пустым.

При нажатии на кнопку «Отмена» состояние графа возвращается на один шаг назад.

При нажатии кнопки «Найти маршрут», программа начинает свою работу по следующему алгоритму:

Программа вызывает getEulerPath(), где сначала идет проверка на связность графа, с помощью функции dfs(), где написан алгоритм поиска в глубину.

Затем программа проверяет степени вершин: если вершин с нечётной степенью нет, то в графе есть эйлеров цикл, если есть 2 вершины с нечётной степенью, то в графе есть только эйлеров путь (эйлерова цикла нет), если же таких вершин больше 2, то в графе нет ни эйлерова цикла, ни эйлерова пути. Чтобы найти эйлеров путь (не цикл), поступим таким образом: если V1 и V2 - это две вершины нечётной степени, то просто добавим ребро (V1,V2), в полученном графе найдём эйлеров цикл (он, очевидно, будет существовать), а затем удалим из ответа "фиктивное" ребро (V1,V2). А после этого можно искать Эйлеров цикл, удалив фиктивное ребро мы получим эйлеров путь.

Поиск Эйлерова цикла:

Начиная со стартовой вершины V строим путь, добавляя на каждом шаге не пройденное еще ребро, смежное с текущей вершиной. Вершины пути накапливаются в стеке S (в моей реализации в качестве такого стека используется стек рекурсии). Когда наступает такой момент, что для текущей вершины X все инцидентные ей ребра уже пройдены, записываем вершины из S в ответ, пока не встретим вершину, которой инцидентны не пройденные еще ребра. Далее продолжаем обход по не посещенным ребрам.

### СКРИНШОТЫ ПРОГРАММЫ

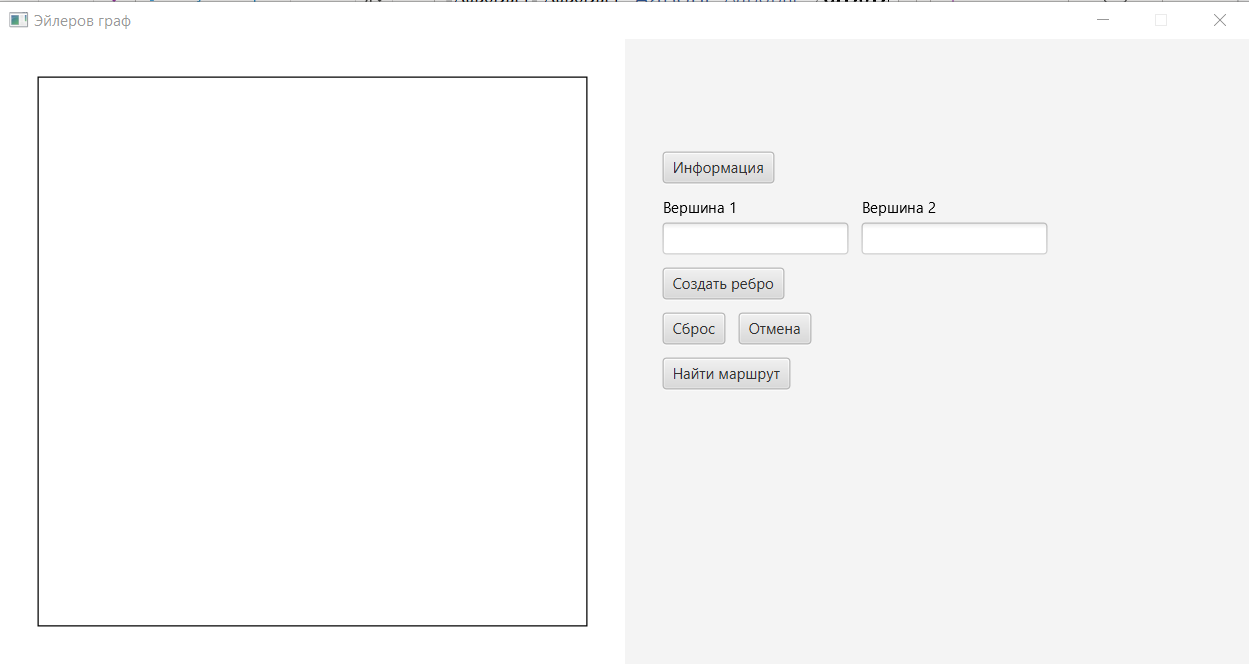


Рис.1 Главное окно

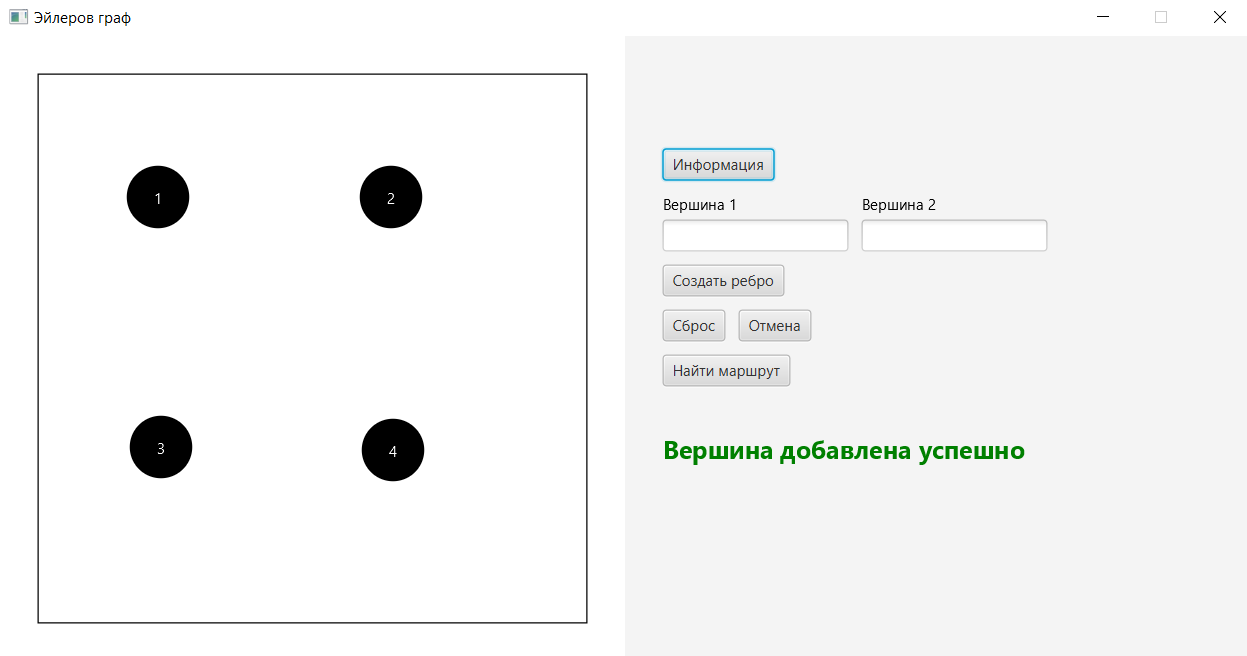


Рис.2 Сообщение об успешном добавлении вершины



Рис.3 Сообщение об успешном добавлении ребра

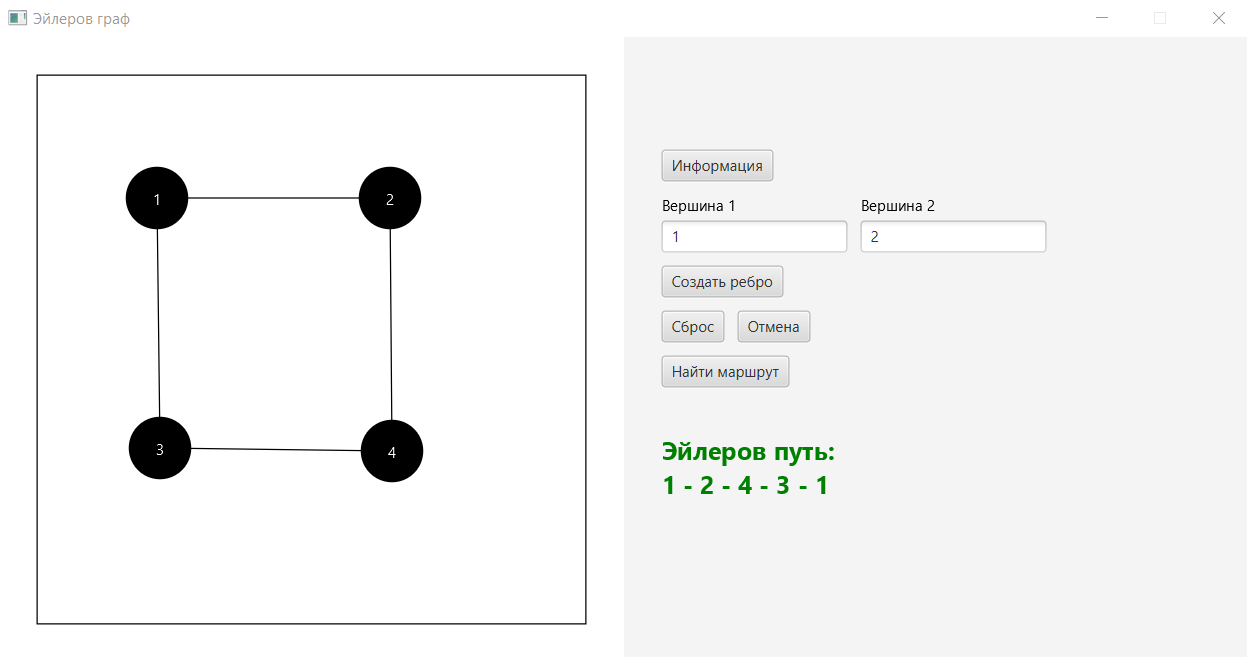


Рис.4 Найденный маршрут

Так как я уже добавляла ребро между 1 и 2 вершиной, если нажать еще раз на кнопку «Создать ребро», ребро не добавится так как будет являться кратным, программа об этом сообщит:

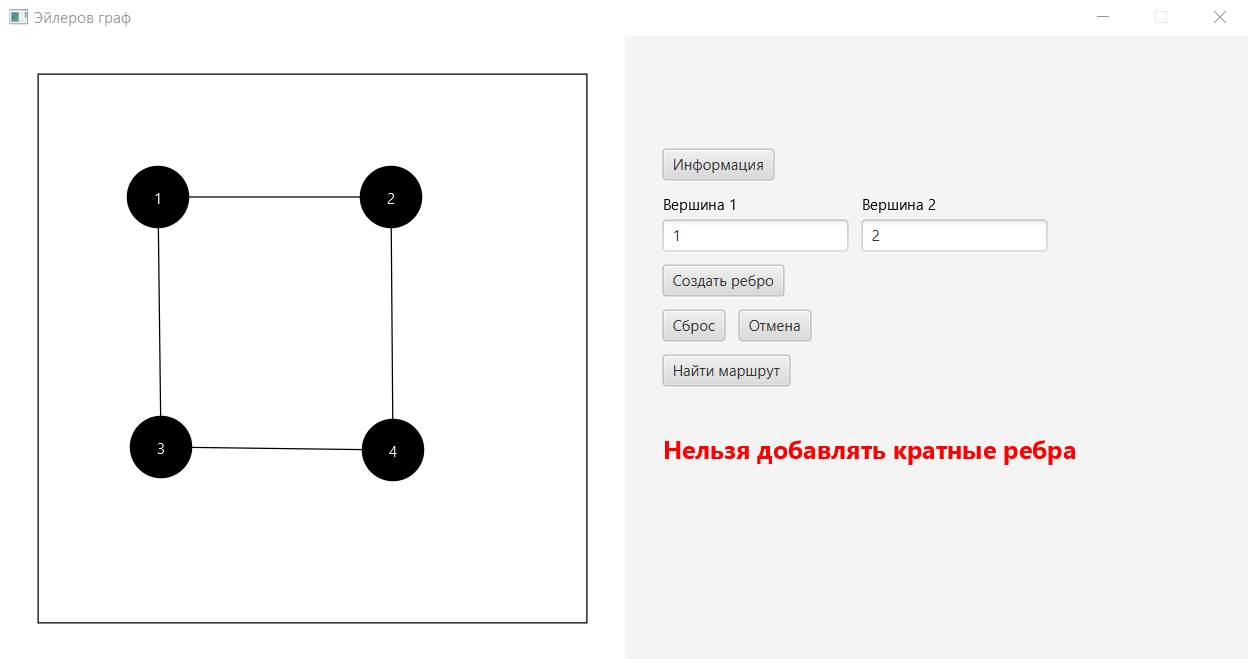


Рис.5 Сообщение о кратном ребре

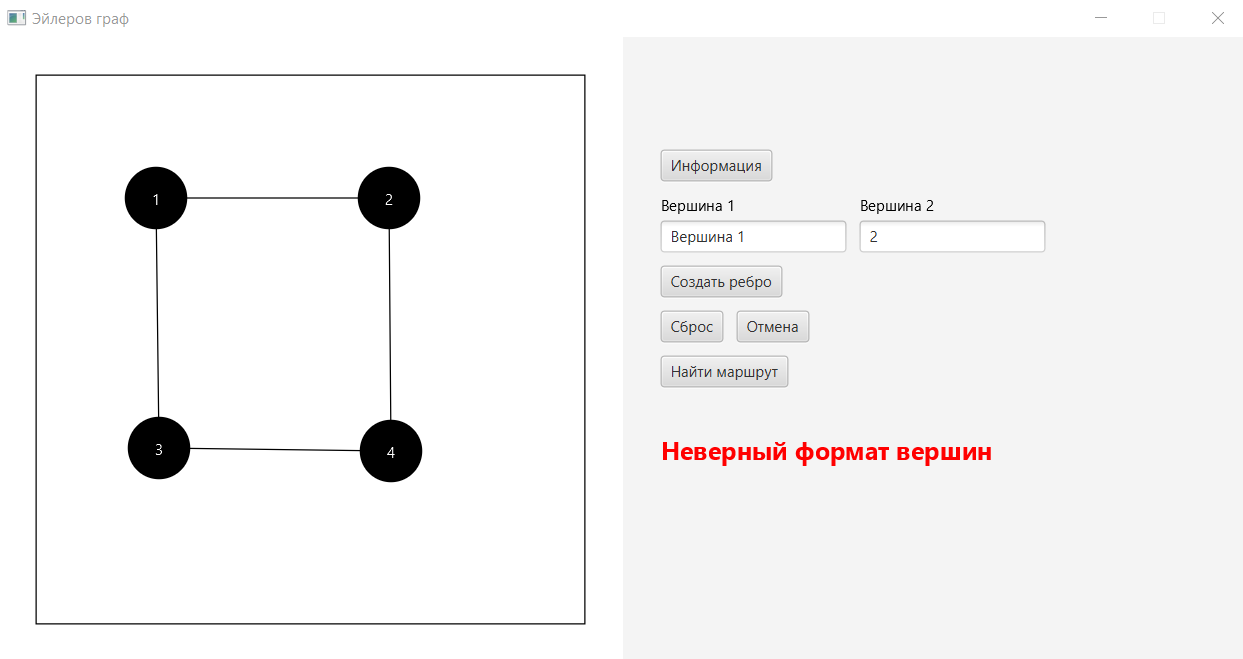


Рис.6 Сообщение о неверном формате вершины, при создании ребра

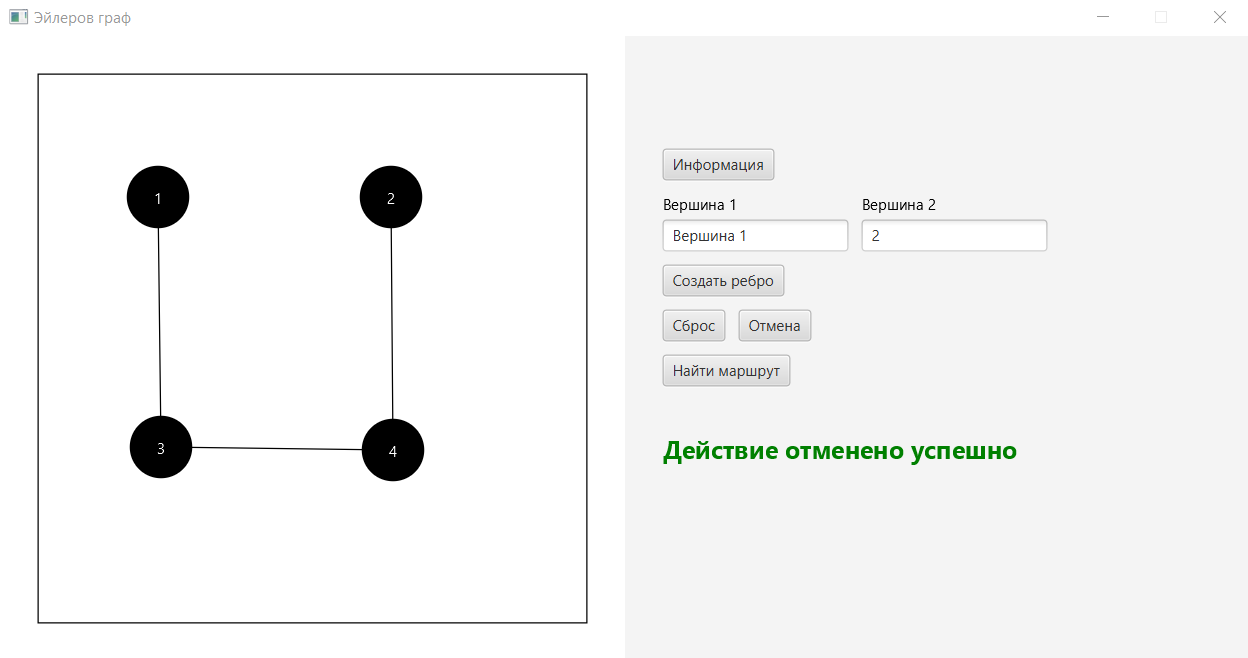


Рис.7 Сообщение об успешной отмене прошлого действия

Если нажать кнопку «Отмена» при пустом поле, также появляется сообщение:

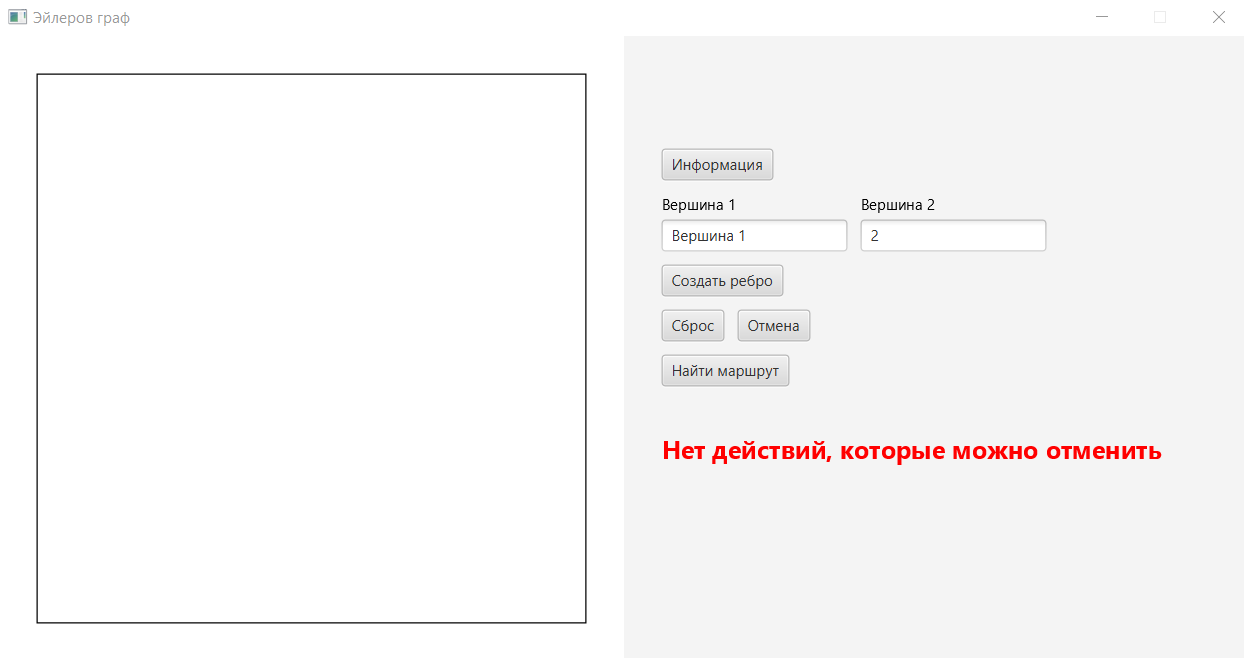


Рис.8 Сообщение об ошибке

### ЛИТЕРАТУРА

1. <https://e-maxx.ru/algo/euler_path#2>
2. <https://ru.wikipedia.org/wiki/Эйлеров_цикл>
3. <https://javarush.ru/groups/posts/605-junit>