

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

ОТЧЕТ
по учебной практике
Тема: Визуализация алгоритма Дейкстры поиска кратчайших путей в
графе.

Студент гр. 2381	_____	Долотов Н.А.
Студент гр. 2381	_____	Богатов И.А.
Студент гр. 2381	_____	Бочаров Г.С.
Руководитель	_____	Фирсов М.А.

Санкт-Петербург
2024

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Долотов Н.А. группы 2381

Студент Богатов И.А. группы 2381

Студент Бочаров Г.С. группы 2381

Тема практики: Командная итеративная разработка пошагового визуализатора алгоритма Дейкстры поиска кратчайших путей в графе с графическим интерфейсом на *Java*.

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на *Java* с графическим интерфейсом.

Алгоритм: алгоритм Дейкстры поиска кратчайших путей в графе.

Сроки прохождения практики: 26.06.2024 – 09.07.2024

Дата сдачи отчета: 09.07.2024

Дата защиты отчета: 09.07.2024

Студент	_____	Долотов Н.А.
Студент	_____	Богатов И.А.
Студент	_____	Бочаров Г.С.
Руководитель	_____	Фирсов М.А.

АННОТАЦИЯ

Целью мини-проекта является овладение новым объектно-ориентированным языком программирования *Java*, а также применение полученных знаний на практике. Задача состоит в реализации визуализатора алгоритма Дейкстры поиска кратчайших путей в графе с графическим интерфейсом.

SUMMARY

The purpose of the mini-project is to master new object-oriented Java programming languages, as well as to apply the knowledge gained in practice. The task is to implement the visualizer of the Dijkstra algorithm for finding shortest paths in a graph with a graphical interface.

СОДЕРЖАНИЕ

	Введение	5
1.	Требования к программе	7
1.1.	Исходные требования к программе	7
1.1.1	Требования к функциональности	7
1.1.2	Требования к визуализации	8
1.2.	Уточнение требований после сдачи прототипа	9
1.3.	Уточнение требований после сдачи 1-ой версии	9
1.4	Уточнение требований после сдачи 2-ой версии	9
2.	План разработки и распределение ролей в бригаде	10
2.1.	План разработки	10
2.2.	Распределение ролей в бригаде	10
3.	Особенности реализации	11
3.1.	Основные классы модели	11
3.2.	Основные классы графического интерфейса	17
4.	Тестирование	21
4.1	Тестирование графического интерфейса	21
4.2	Тестирование кода графа	22
4.3	Тестирование кода алгоритма	23
4.4	Тестирование кода сохранения графа	24
	Заключение	25
	Список использованных источников	26

ВВЕДЕНИЕ

Основная цель – реализация визуализатора алгоритма Дейкстры поиска кратчайших путей в графе в виде приложения с графическим интерфейсом. Для её достижения требуется изучить основные понятия объектно-ориентированного языка программирования *Java*, составить спецификацию и план разработки, распределить обязанности в бригаде для выполнения итеративной разработки проекта.

Алгоритм Дейкстры используется для поиска кратчайших путей в графе. Он работает путем постепенного расширения множества вершин, для которых известен кратчайший путь от начальной вершины, выбора вершины с минимальным известным расстоянием и обновления расстояний до ее соседей. Этот процесс продолжается, пока не будут найдены кратчайшие пути до всех вершин в графе.

Псевдокод:

```
// V - множество вершин графа
// E - множество рёбер графа
// start - начальная вершина
// dist[] - массив минимальных расстояний от начальной вершины
// used[] - массив обработанных вершин
// w(i) - вес i-го ребра
// e.to - вершина, в которую ведёт ребро e от текущей вершины v

function dijkstra(start):
    for v ∈ V:
        dist[v] = ∞
        used[v] = false
    dist[start] = 0
    for i ∈ V:
        v = null
        for j ∈ V:
            if !used[j] and (v == null or dist[j] < dist[v]):
                v = j
        if dist[v] = ∞:
            break
        used[v] = true
        for e ∈ E (e - edge coming from vertex v):
            if dist[v] + w(e) < dist[e.to]:
                dist[e.to] = dist[v] + w(e)
```

Алгоритм Дейкстры используется в следующих системах:

- Сетевые маршрутизаторы: определение оптимальных маршрутов для передачи данных в компьютерных сетях;
- Географические информационные системы: нахождение кратчайших путей на картах (навигация);
- Игровая разработка: определение путей для персонажей и объектов в игровых мирах;
- Логистика и транспорт: оптимизация маршрутов доставки и передвижения транспорта.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1 Исходные требования к программе

1.1.1 Требования к функциональности

1. Построение графа в среде приложения на холсте:
 - 1.1. Режим *“Edit Mode”*:
 - 1.1.1. Добавление вершины – клик ЛКМ;
 - 1.1.2. Перемещение вершины – зажатие ЛКМ на вершине и перемещение мыши;
 - 1.1.3. Добавление ребра – два последовательных клика ЛКМ по вершинам;
 - 1.1.4. Задание веса ребру – двойной клик ЛКМ по ребру и ввод веса.
 - 1.2. Режим *“Delete Mode”*: удаление вершины/ребра – клик ЛКМ по вершине/ребру.
 - 1.3. Действие *“Clear the Field”*: очистка холста.
 - 1.4. Действие *“Switch Type Graph”*: изменение ориентации графа (ориентированный/неориентированный).
2. Загрузка/сохранение графа из файла *.json*:
 - 2.1. Действие *“Load the Graph”*: загрузка графа.
 - 2.2. Действие *“Save the Graph”*: сохранение графа.
3. Выполнение алгоритма:
 - 3.1. Действие *“Run Completely”*: запуск алгоритма с начальной вершины без пошаговой визуализации.
 - 3.2. Действие *“Step Back”*: возврат к предыдущему шагу работы алгоритма.
 - 3.3. Действие *“Step Forward”*: переход к следующему шагу работы алгоритма.

1.1.2 Требования к визуализации

Визуализация работы алгоритма непосредственно на графе:

1. Выбранная начальная вершина графа обозначена цветом;
2. Текущая обрабатываемая вершина выделена цветом;
3. Текущее обрабатываемое ребро выделена цветом;
4. Текущая вершина-сосед выделена цветом;
5. Текущее новое расстояние отображается рядом с вершиной в виде неравенства (сравнение нового расстояния с текущим минимальным);
6. Минимальные расстояния отображены рядом с каждой вершиной.

Текстовые пояснения визуализации отображены в области вывода шагов алгоритма.

Запустить	Шаг назад	Шаг вперёд	Строить	Удалить	Импорт	Экспорт
Шаги работа алгоритма			Поле для построения и визуализации графа			

Рисунок 1 – схема графического интерфейса приложения

1.2 Уточнение требований после сдачи прототипа

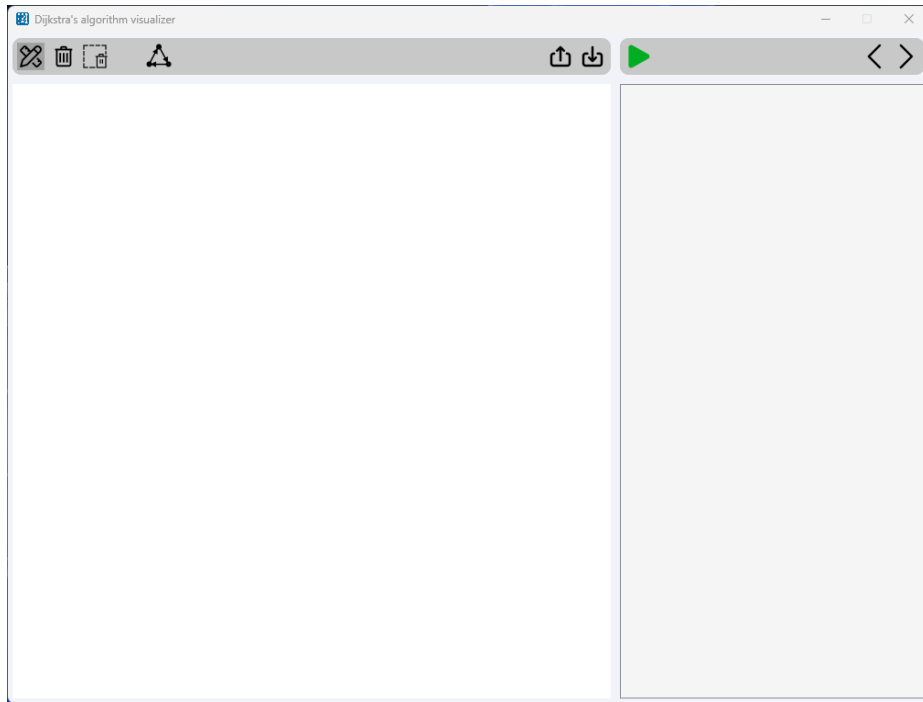


Рисунок 2 – графический интерфейс приложения после сдачи прототипа

1.3 Уточнение требований после сдачи 1-ой версии

- Режим “*Edit Mode*”: задание веса ребру – клик ПКМ по ребру и ввод веса;
- Окно ввода веса ребра должно поддерживать нажатие на клавиатуре “*Enter*” для подтверждения;
- Номера вершин должны выводиться поверх рёбер.

1.4 Уточнение требований после сдачи 2-ой версии

- Стартовая вершина должна сохраняться/загружаться из файла вместе с графом;
- При попытке запустить алгоритм без стартовой вершины требуется сообщить о том, что нужно её выбрать;
- Увеличить размер шрифта в области вывода текстовых пояснений.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Дата	Этап проекта	Реализованные возможности	Выполнено
28.06.2024	Согласование спецификации	-	-
01.07.2024	Сдача прототипа	Графический интерфейс без функциональных возможностей	Запланированный функционал
03.07.2024	Сдача версии 1	Построение графа на холсте; запуск алгоритма без пошаговой визуализации; вывод текстовых пояснений работы алгоритма	Запланированный функционал
06.07.2024	Сдача версии 2	Пошаговое выполнение алгоритма; пошаговая визуализация; загрузка/сохранение графа в файл	Запланированный функционал
08.07.2024	Сдача версии 3	Внесение правок	Запланированный функционал
08.07.2024	Сдача отчёта	-	-
09.07.2024	Защита отчёта	-	-

2.2. Распределение ролей в бригаде

- Долотов Никита – API графа, реализация алгоритма, интерфейс.
- Богатов Илья – Визуализация алгоритма, тестирование, оформление отчёта.
- Бочаров Глеб – Модификация алгоритма поиска в двух направлениях.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Основные классы модели

Класс *Vertex*.

Содержит приватные поля: идентификатор вершины *id*, строковая метка *label*, координаты вершины *x* и *y*, цвет *color*.

Реализованы сеттеры и геттеры полей.

Реализован публичный метод *toJSON()*, преобразующий объект вершины в *JSON*-строку и возвращает её.

Реализован публичный метод *fromJSON()*, принимающий строку *json* и создающий из неё объект вершины.

Класс *Edge*.

Содержит приватные поля: начальная вершина *fromV*, конечная вершина *toV*, вес *weight*, цвет *color*.

Реализованы сеттеры и геттеры полей.

Реализован публичный метод *toJSON()*; преобразует объект ребра в *JSON*-строку и возвращает её.

Реализован публичный метод *fromJSON()*, принимающий граф *graph*, строку *json*; создает из *json* объект вершины.

Класс *DirectedGraph*.

Содержит приватные поля: словарь вершин по идентификаторам *vertices*, словарь исходящих ребер по вершинам (список смежности) *adjacencyList*, флаг ориентированности графа *isDirected*, метка для следующей вершины *nextLabel*.

Реализован публичный метод *isDirected()*; возвращает флаг, указывающий, является ли граф ориентированным.

Реализован публичный метод *addVertex()*, принимающий вершину *vertex*; добавляет *vertex* в граф, присваивая метку.

Реализован публичный метод *addEdge()*, принимающий ребро *edge*; добавляет *edge* в граф, если такого ребра ещё нет.

Реализован публичный метод *removeVertex()*, принимающий вершину *vertex*; удаляет *vertex* и все рёбра, инцидентные ей, из графа.

Реализован публичный метод *removeEdge()*, принимающий ребро *edge*; удаляет *edge* из графа.

Реализован публичный метод *clear()*; очищает граф, удаляя все вершины и рёбра.

Реализован публичный метод *setEdgeWeight()*, принимающий ребро *edge*, вес *weight*; устанавливает вес *weight* ребру *edge*.

Реализован публичный метод *getVertices()*; возвращает список всех вершин в графе.

Реализован публичный метод *getEdgesFrom()*, принимающий вершину *vertex*; возвращает список всех рёбер, исходящих из *vertex*.

Реализован публичный метод *getVertexById()*, принимающий идентификатор вершины *id*; возвращает вершину по её *id*.

Реализован публичный метод *getVertexByColor()*, принимающий цвет *color*; возвращает вершину по её *color*, если такая существует.

Реализован публичный метод *toJSON()*; преобразует объект ориентированного графа в *JSON*-строку, включая информацию о вершинах и рёбрах, и возвращает её.

Реализован публичный метод *fromJSON()*, принимающий строку *json*; создает из *json* объект ориентированного графа, восстанавливая вершины и рёбра из *JSON*-данных.

Реализован приватный метод *reassignLabels()*; переназначает метки вершинам после удаления вершины.

Класс *UndirectedGraph*.

Наследуется от *DirectedGraph*.

Переопределен публичный метод *addEdge()*, принимающий ребро *edge*; добавляет *edge* в граф и автоматически добавляет обратное ребро, чтобы сохранить неориентированность.

Переопределен публичный метод *removeEdge()*, принимающий ребро *edge*; удаляет *edge* и соответствующее ему обратное ребро.

Переопределен публичный метод *setEdgeWeight()*, принимающий ребро *edge*, вес *weight*; устанавливает вес *weight* ребра *edge* и соответствующего ему обратного ребра.

Переопределен публичный метод *fromJSON()*, принимающий строку *json*; создает из *json* объект неориентированного графа, восстанавливая вершины и рёбра из *JSON*-данных.

Класс *DijkstraState*.

Содержит приватные поля: словарь расстояний по вершинам *distances*, множество посещённых вершин *visited*, список шагов *steps*, текущую вершину *currentVertex*, текущее ребро *currentEdge*, соседнюю вершину *neighborVertex* и строку неравенства *inequality*.

Реализован конструктор и геттеры полей.

Класс *DijkstraAlgorithm*.

Абстрактный класс алгоритма Дейкстры.

Содержит 4 абстрактных метода.

Публичный метод *process()* принимает на вход 2 вершины и запускает алгоритм.

Защищенный метод *saveState* принимающий текущую вершину *vertex*, текущее ребро *edge*, вершину-соседа *neighbor*, строку неравенства *inequality*; сохраняет текущее состояние алгоритма, включая словарь расстояний по вершинам, множество посещённых вершин, список шагов, текущую вершину, текущее ребро, соседнюю вершину и неравенство.

Публичный метод *getState()*, принимающий индекс *index*; возвращает состояние алгоритма по *index*.

Приватный метод *saveState()*, принимающий текущую вершину *vertex*, текущее ребро *edge*, вершину-соседа *neighbor*, строку неравенства *inequality*;

сохраняет текущее состояние алгоритма, включая словарь расстояний по вершинам, множество посещённых вершин, список шагов, текущую вершину, текущее ребро, соседнюю вершину и неравенство.

Класс *DijkstraAlgorithm1*.

Класс алгоритма Дейкстры с модификацией поиска в 2-х направлениях.

Наследуется от класса *DijkstraAlgorithm*.

Содержит приватные поля: граф *graph*, словари расстояний по вершинам при прямом и обратном обходе *forward_distances*, *backward_distances*, множество посещённых при прямом и обратном обходе вершин *forward_visited*, *backward_visited*, приоритетные очереди для обоих обходов *forward_queue*, *backward_queue*, словари путей *forward_paths* и *backward_paths* для восстановления пути, список шагов *steps*, список состояний *states*.

Реализован публичный метод *process()*, принимающий вершины *start* и *end*; запускает алгоритм Дейкстры, начиная с указанных начальной и конечной вершин; обрабатывает вершины, обновляет расстояния и сохраняет состояния на каждом шаге. Поиск ведется в двух направлениях: от начала по прямым ребра и от конца по обратным ребрам. Поиск прекращается при отсутствии пути или после двойной обработки одной и той же вершины.

Реализован публичный метод *getState()*, принимающий индекс *index*; возвращает состояние алгоритма по *index*.

Реализован публичный метод *getNumberStates()*, возвращает количество сохранённых состояний алгоритма.

Реализован приватный метод *saveState()*, принимающий текущую вершину *vertex*, текущее ребро *edge*, вершину-соседа *neighbor*, строку неравенства *inequality*; сохраняет текущее состояние алгоритма, включая словарь расстояний по вершинам, множество посещённых вершин, список шагов, текущую вершину, текущее ребро, соседнюю вершину и неравенство.

Класс *DijkstraAlgorithm2*.

Класс алгоритма Дейкстры без модификаций.

Наследуется от класса *DijkstraAlgorithm*.

Содержит приватные поля: граф *graph*, словарь расстояний по вершинам *distances*, множество посещённых вершин *visited*, приоритетная очередь вершин *queue*, список шагов *steps*, список состояний *states*.

Реализован публичный метод *process()*, принимающий вершину *start*; запускает алгоритм Дейкстры, начиная с указанной стартовой вершины; обрабатывает вершины, обновляет расстояния и сохраняет состояния на каждом шаге.

Реализован публичный метод *getState()*, принимающий индекс *index*; возвращает состояние алгоритма по *index*.

Реализован публичный метод *getNumberStates()*, возвращает количество сохранённых состояний алгоритма.

Реализован приватный метод *saveState()*, принимающий текущую вершину *vertex*, текущее ребро *edge*, вершину-соседа *neighbor*, строку неравенства *inequality*; сохраняет текущее состояние алгоритма, включая словарь расстояний по вершинам, множество посещённых вершин, список шагов, текущую вершину, текущее ребро, соседнюю вершину и неравенство.

Класс *AlgorithmManager*.

Содержит приватные поля: объект приложения *app*, объект алгоритма *DijkstraAlgorithm dijkstra*, флаг выполнения алгоритма *isRun* и индекс текущего состояния *stateIndex*.

Реализован публичный метод *getState()*; возвращает текущее состояние алгоритма Дейкстры по индексу *stateIndex*.

Реализован публичный метод *isRun()*; возвращает флаг выполнения алгоритма *isRun*.

Реализован публичный метод *reset()*; сбрасывает алгоритм и граф в исходное состояние, обнуляет флаг выполнения и индекс состояния, устанавливает исходные цвета для вершин и рёбер.

Реализован публичный метод *stepBack()*; выполняет шаг назад по состояниям алгоритма.

Реализован публичный метод *stepForward()*; выполняет шаг вперёд по состояниям алгоритма

Реализован публичный метод *runFull()*; полностью выполняет алгоритм, переходя к последнему шагу.

Реализован приватный метод *run()*; запускает выполнение алгоритма Дейкстры, начиная с выбранной стартовой вершины, сохраняя состояния.

Реализован приватный метод *update()*; обновляет текстовое пояснение шагов алгоритма и перерисовывает холст с графом.

Реализован приватный метод *setAlgoState()*; обновляет поле отвечающее за тип алгоритма: простой или с модификацией.

3.2. Основные классы графического интерфейса

Класс *ControlPanelsManager*.

Содержит приватные поля: объект приложения *app*, объект *constraints* для размещения компонентов *gbc*, объект *ButtonsManager buttonsManager*, кнопки *editButton*, *deleteButton* и *switchGraphTypeButton*.

Реализован конструктор *ControlPanelsManager()*, принимающий объект приложения *app*; инициализирует менеджер панелей управления с указанным приложением; создает кнопки и панели управления.

Реализован публичный метод *getEditButton()*; возвращает кнопку переключения режима редактирования.

Реализован публичный метод *getDeleteButton()*; возвращает кнопку переключения режима удаления.

Реализован приватный метод *initControlPanelLeft()*; инициализирует левую панель управления, добавляя кнопки редактирования, удаления, переключения типа графа, сохранения и загрузки графа, а также добавляет обработчики событий для этих кнопок.

Реализован приватный метод *initControlPanelRight()*; инициализирует правую панель управления, добавляя кнопки запуска алгоритма, шага назад и шага вперёд, а также добавляет обработчики событий для этих кнопок.

Реализован приватный метод *switchAlgoType()*; для переключения между типами алгоритма.

Реализован приватный метод *deleteAll()*; очищает холст графа, удаляет все вершины и рёбра, сбрасывает алгоритм и перерисовывает холст.

Реализован приватный метод *switchGraphType()*;

Реализован приватный метод *switchGraphType()*; переключает тип графа (ориентированный/неориентированный), очищает граф, сбрасывает алгоритм и обновляет иконку кнопки переключения типа графа.

Реализован приватный метод *save()*; сохраняет текущий граф в *JSON*-файл через диалоговое окно сохранения файла.

Реализован приватный метод *load()*; загружает граф из *JSON*-файла через диалоговое окно открытия файла, устанавливает загруженный граф в приложении и обновляет отображение.

Реализован приватный метод *updateToggleButtonStyles()*, принимающий группу кнопок-переключателей *toggleGroup*; обновляет стили кнопок-переключателей в зависимости от их состояния (выбрана/не выбрана).

Реализован приватный метод *changeSwitchGraphTypeButtonIcon()*, принимающий кнопку *button*, флаг ориентированности графа *isDirected*; обновляет иконку и подсказку кнопки переключения типа графа в зависимости от текущего типа графа.

Реализован вложенный приватный класс *RoundedPanel*, наследуемый от *JPanel*; реализует закругленные панели с переопределенным методом *paintComponent()* для отрисовки панели с закругленными углами.

Класс *GraphFieldManager*.

Содержит приватные поля: объект приложения *app*, панель для отображения графа *graphField*, выбранную вершину *selectedVertex*, первую вершину для добавления ребра *firstVertex* и точку начального клика мыши *initClick*.

Реализован конструктор *GraphFieldManager()*, принимающий объект приложения *app*; инициализирует менеджер поля графа с указанным приложением; создает панель для графа, добавляет слушатели мыши и настраивает отображение панели.

Реализован публичный метод *getGraphField()*; возвращает панель для отображения графа.

Реализован публичный метод *setFirstVertex()*, принимающий первую вершину *vertex*; устанавливает первую вершину для добавления ребра.

Реализован публичный метод *getFirstVertex()*; возвращает первую вершину для добавления ребра.

Реализован публичный метод *reset()*; сбрасывает выбранную вершину, первую вершину для добавления ребра и точку начального клика мыши.

Реализован приватный метод *addVertex()*, принимающий точку *point*; добавляет вершину в граф по *point*, обновляет алгоритм и перерисовывает графическое поле.

Реализован приватный метод *addEdge()*, принимающий начальную *from* и конечную *to* вершину; добавляет ребро в граф между *from* и *to*, обновляет алгоритм и перерисовывает графическое поле.

Реализован приватный метод *removeVertex()*, принимающий вершину *vertex*; удаляет *vertex* из графа, обновляет алгоритм и перерисовывает графическое поле.

Реализован приватный метод *removeEdge()*, принимающий ребро *edge*; удаляет *edge* из графа, обновляет алгоритм и перерисовывает графическое поле.

Реализован приватный метод *getVertexAt()*, принимающий точку *point*; возвращает вершину, находящуюся в *point*, или *null*, если вершина не найдена.

Реализован приватный метод *getEdgeAt()*, принимающий точку *point*; возвращает ребро, находящееся в *point*, или *null*, если ребро не найдено.

Реализован приватный метод *isPointOnLine(Point point, Point from, Point to)*, принимающий точки *point*, *from*, *to*; проверяет, находится ли *point* на линии между *from* и *to*.

Реализован вложенный приватный класс *GraphPainter*, наследуемый от *JPanel* (отрисовка графа); переопределяет метод *paintComponent()* для отрисовки вершин, рёбер и текущего состояния алгоритма Дейкстры.

Реализован вложенный приватный класс *MouseListener*, наследуемый от *MouseAdapter*; переопределяет слушатель событий мыши для обработки кликов и нажатий.

Реализован вложенный приватный класс *MouseMotionListener*, наследуемый от *MouseMotionAdapter*; переопределяет слушатель событий движения мыши для обработки перетаскивания вершин.

Класс *StepsFieldManager*.

Содержит приватные поля: объект приложения *app* и текстовую область *stepsField* для отображения шагов алгоритма.

Реализован конструктор *StepsFieldManager()*; инициализирует менеджер поля шагов с указанным приложением; создаёт текстовую область *stepsField*, настраивает её внешний вид и прокручиваемую панель для неё, а также добавляет эту панель в интерфейс приложения.

Реализован публичный метод *display()*; отображает текущие шаги алгоритма Дейкстры в текстовой области.

Реализован публичный метод *clear()*; очищает текстовую область шагов.

Класс *App*.

Содержит приватные поля: объект *constraints* для размещения компонентов *gbc*, объект графа *graph*, и объекты менеджеров интерфейса *controlPanelsManager*, *graphFieldManager*, *stepsFieldManager* и *algorithmManager*.

Реализован сеттер графа *graph* и геттеры объекта *constraints* для размещения компонентов, графа, менеджер панелей управления, менеджер холста графа, менеджер поля шагов, менеджер алгоритмов.

Реализован конструктор *App()*; инициализирует приложение; настраивает основные параметры окна, инициализирует компоненты интерфейса и добавляет их в окно.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

Было произведено ручное тестирование на двух операционных системах (*Linux Ubuntu* и *Windows*) всех элементов интерфейса, а именно:

- Режима изменения графа;
- Режима удаления графа;
- Воспроизведение алгоритма Дейкстры;
- Панели инструментов (*Clear the Field, Switch Graph Type*);
- Инструментов сохранения и загрузки графа;
- Окна логирования.

Во время тестирования режима изменения графа была проверена корректная работоспособность каждого инструмента: перетаскивание вершин графа, создание вершины, создание ребра, установка веса ребра, выбор начальной вершины для алгоритма Дейкстры, выбор конечной вершины для алгоритма Дейкстры, очистка графа. Инструменты работают исправно, конфликта не возникает. При выборе инструмента, который отвечает за редактирование графа, графически подсвечивается какая кнопка была выбрана пользователем, при этом невозможно выбрать одновременно два режима, только один.

Во время тестирования воспроизведения алгоритма Дейкстры также была проверена работоспособность каждого инструмента: следующий/предыдущий шаг, полный запуск. Если пользователь не выбрал начальную вершину – появляется окно с предупреждением о том, что требуется выбрать начальную вершину. Были протестированы ситуации, когда пользователь, во время проигрывания алгоритма Дейкстры, начинает менять настройки или как-либо взаимодействовать с инструментами, которые ему доступны, никаких конфликтов не возникает, в случае изменения графа или начальной вершины алгоритм прекращает свою работу, пользователю необходимо заново его

начать, так как он изменил какие-либо настройки. В случае, когда настройки не были изменены – алгоритм продолжает воспроизводиться.

Инструменты сохранения и загрузки работают корректно. При сохранении графа с помощью инструмента “*Save the Graph*” – будет открыт проводник, где нужно выбрать путь сохранения. Загрузка происходит при помощи инструмента “*Load the Graph*”. Расширение у сохраняемых файлов будет *.json*. Инструмент загрузки файлов будет загружать только целые файлы формата *.json*, то есть, если файл был поврежден каким-либо образом – приложение не сможет его загрузить, конфликта не возникнет.

Окно логирования не доступно для редактирования пользователем, что было протестировано. Любое действие пользователя записывается в окно логирования.

4.2. Тестирования кода графа

Тестирования реализации графа подразумевает под собой создание двух групп автоматического тестирования для каждого вида графа, т.к. реализация некоторых функций у разных типов разная.

Для тестирования использовалась библиотека *JUnit*, которая сильно упростила этот процесс. Удобной возможностью оказалась функция, которая вызывается перед каждым тестом, в которой производился сброс графа.

Каждая функция в обоих видах тестировалась минимум тремя тестами: тестом на нормальную работу, на получение ошибки при передаче *null* значения или на получения ошибки при других обстоятельствах.

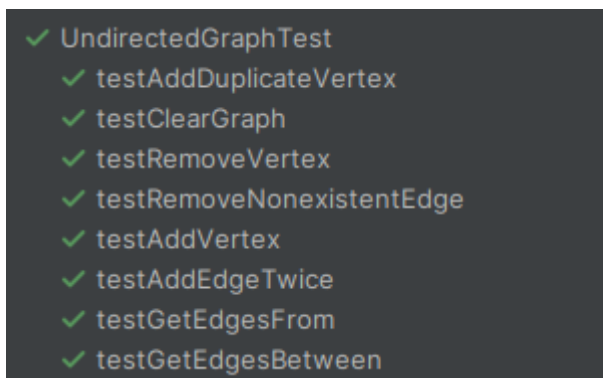


Рисунок 3 – Тесты для неориентированного графа

✓ DirectedGraphTest	6 ms
✓ testAddEdge	1 ms
✓ testClearGraph	1 ms
✓ testRemoveVertex	3 ms
✓ testRemoveEdge	0 ms
✓ testAddVertex	1 ms

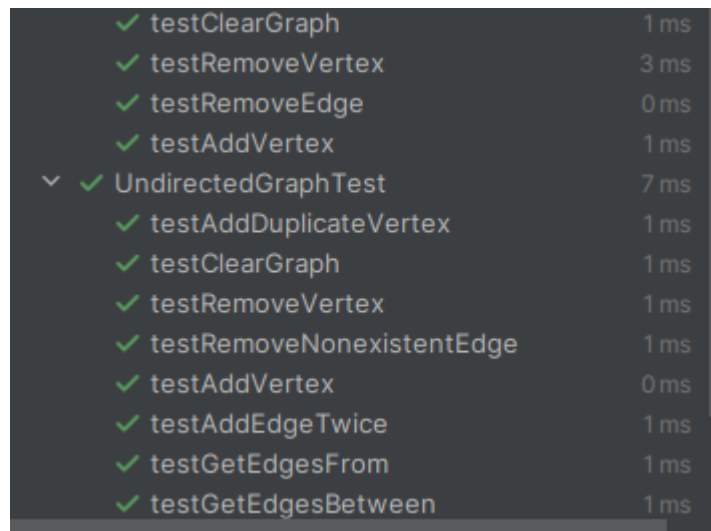
Рисунок 4 – Тесты для ориентированного графа

4.3. Тестирование кода алгоритма

Для тестирования кода алгоритма были созданы несколько автоматических тестов с помощью библиотеки *JUnit*. С её помощью тестировался сам алгоритм на ориентированном и неориентированном графе, а также ситуации, когда путь не существует, либо же когда существуют несколько кратчайших путей. Сами тесты подбирались, чтобы охватывать все описанные случаи.

✓ testClearGraph	1 ms
✓ testRemoveVertex	3 ms
✓ testRemoveEdge	0 ms
✓ testAddVertex	1 ms
✓ UndirectedGraphTest	7 ms
✓ testAddDuplicateVertex	1 ms
✓ testClearGraph	1 ms
✓ testRemoveVertex	1 ms
✓ testRemoveNonexistentEdge	1 ms
✓ testAddVertex	0 ms
✓ testAddEdgeTwice	1 ms
✓ testGetEdgesFrom	1 ms
✓ testGetEdgesBetween	1 ms

Рисунок 5 – Тестирование алгоритма на неориентированном графе

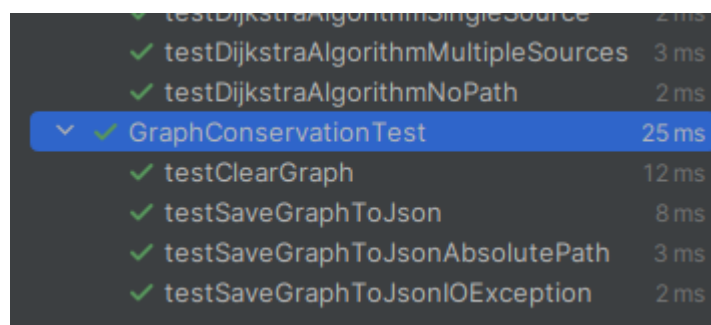


✓ testClearGraph	1 ms
✓ testRemoveVertex	3 ms
✓ testRemoveEdge	0 ms
✓ testAddVertex	1 ms
✓ UndirectedGraphTest	7 ms
✓ testAddDuplicateVertex	1 ms
✓ testClearGraph	1 ms
✓ testRemoveVertex	1 ms
✓ testRemoveNonexistentEdge	1 ms
✓ testAddVertex	0 ms
✓ testAddEdgeTwice	1 ms
✓ testGetEdgesFrom	1 ms
✓ testGetEdgesBetween	1 ms

Рисунок 6 – Тестирование алгоритма на ориентированном графе

4.4. Тестирование кода сохранения графа

Для тестирования кода сохранения графа были созданы несколько автоматических тестов с помощью библиотеки *JUnit*. С её помощью тестировалась как загрузка, так и записи графа в текстовый файл формата *JSON*, причем сохранение тестировалось в разных директориях, как относительно запущенной программы, так и абсолютно начиная с дисков, а также ситуации, если невозможно открыть файл. Сами тесты были подобраны таким образом, чтобы охватить все описанные случаи, причем над ними была проведена дополнительная настройка, чтобы порядок выполнения тестов соответствовал их порядку в коде, так как некоторые из них взаимосвязаны. К тому же, после выполнения тестов все созданные в тестах файлы с графами автоматически удаляются с компьютера.



✓ testDijkstraAlgorithmSingleSource	2 ms
✓ testDijkstraAlgorithmMultipleSources	3 ms
✓ testDijkstraAlgorithmNoPath	2 ms
✓ GraphConservationTest	25 ms
✓ testClearGraph	12 ms
✓ testSaveGraphToJson	8 ms
✓ testSaveGraphToJsonAbsolutePath	3 ms
✓ testSaveGraphToJsonIOException	2 ms

Рисунок 7 – Тестирование сохранения графа

ЗАКЛЮЧЕНИЕ

В ходе выполнения мини-проекта было реализовано приложение с графическим интерфейсом, содержащее графический редактор графов, демонстрирующее пошаговое выполнение алгоритма Дейкстры поиска кратчайших путей в графе и его модификации. Получены навыки программирования на объектно-ориентированном языке программирования *Java*.

Разработанное приложение соответствует требованиям, предъявленным в начале работы, а также уточнениям, которые были получены в процессе итеративной разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Репозиторий бригады // *GitHub*. URL: https://github.com/Do1otov/Dijkstra_Algo_Visualization.
2. Система вопросов и ответов о программировании // *Stack Overflow*. URL: <https://stackoverflow.com/> (дата обращения: 30.06.2024, 01.07.2024, 05.07.2024).
3. Стоковая графика для дизайна // *Icons8*. URL: <https://icons8.ru/> (дата обращения: 30.06.2024).
4. Википедия конспектов // *neerc.ifmo*. 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 (дата обращения: 01.07.2024).
5. Платформа для изучения программирования на *Java* // *JavaRush*. URL: <https://javarush.com/> (дата обращения: 02.07.2024, 05.07.2024).
6. Платформа для изучения программирования на *Java* // *Progoschool*. URL: <https://progoschool.ru/java/java-swing/> (дата обращения: 30.06.2024).