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

ОТЧЕТ

по учебной практике

Тема: Кратчайшие пути в графе. Алгоритм Дейкстры

Студент гр. 0304 _____ Никитин Д.Э.

Студент гр. 0304 _____ Жиглов Д.С.

Студент гр. 0304 _____ Нагибин И.С.

Руководитель _____ Фирсов М.А.

Санкт-Петербург
2022

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Никитин Д.Э. группы 0304

Студент Жиглов Д.С. группы 0304

Студент Нагибин И.С. группы 0304

Тема практики: Кратчайшие пути в графе. Алгоритм Дейкстры

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Kotlin с графическим интерфейсом.

Алгоритм: Дейкстра.

Сроки прохождения практики: 29.06.2022 – 12.07.2022

Дата сдачи отчета: 01.07.2022

Дата защиты отчета: 12.07.2022

Студент _____ Никитин Д.Э.

Студент _____ Жиглов Д.С.

Студент _____ Нагибин И.С.

Руководитель _____ Фирсов М.А.

АННОТАЦИЯ

Необходимо разработать визуализатор алгоритма на графе, который выполняется пошагово с пояснениями. В качестве алгоритма выбран алгоритм Дейкстры, поскольку он имеет прикладное значение в протоколах маршрутизации OSPF и IS-IS.

Данная практическая работа нацелена на формирование у студентов: умения работать в команде, самодисциплины и навыка оценки трудоёмкости задач.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью практического задания является разработка пошагового визуализатора алгоритма на графе. В качестве алгоритма выбран алгоритм Дейкстры.

Алгоритм Дейкстры применяется для поиска кратчайших путей от исходной вершины графа до любой. Применяется данный алгоритм в протоколах маршрутизации OSPF и IS-IS.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

Необходимо разработать пошаговый визуализатор алгоритма на графе, позволяющий выполнить следующие функции:

- ввод данных пользователем за счёт графического интерфейса / из файла
- визуализация введённых данных в виде графа.
- пошаговое применение алгоритма Дейкстры для нахождения кратчайшего пути от выбранной начальной вершины до каждой вершины графа с возможностью возврата к предыдущему шагу(машина состояний). Отображение весов путей над вершинами, вывод пояснений в отдельном окне на каждом шаге.

1.1.1. Требования к визуализации

Рисунок 1 - Интерфейс программы

Интерфейс должен быть интуитивно понятным для конечного пользователя. Для этого необходимо выполнить разделение на 3 области: с возможностью выбора действия(назовем её панель инструментов), область с

отрисовкой графа(назовем холст), а также с выводом пояснений(информационная панель).

Панель инструментов должна состоять из 11 кнопок со следующими опциями:

- Загрузка
- Сохранение
- Добавить вершину
- Удалить вершину
- Добавить ребро
- Удалить ребро
- Запуск алгоритма
- Шаг вперёд алгоритма
- Шаг назад алгоритма
- Пошаговое воспроизведение алгоритма
- Переход к финальному шагу алгоритма

Холст — это layout типа Box, который не имеет фиксированных границ и используется для отрисовки вершин и ребер. Для перемещения по холсту используется мышь: задерживая левую кнопку мыши появляется возможность перемещаться по холсту. При прокрутке колеса произойдет ожидаемое приближение или отдаление холста.

Граф состоит из 2-ух множеств: вершин и ребер. Вершины и ребра имеют разделение на логическую часть и отрисовку. Отрисовка представляет собой поверхность определенной формы с возможность нажатия на неё. При нажатии на поверхность и выбранном инструменте происходит определенное действие(например: при нажатии на две различные вершины с выбранным инструментом «добавить ребро» создается ребро в графе и происходит его отрисовка).

После завершения формирования графа необходимо нажать кнопку старта и нажать на вершину, относительно которой алгоритм Дейкстры начнёт работу. Над всеми вершинами отобразится бесконечный вес. На следующем

шаге вершина загорится зеленым цветом и обновит вес на 0. Окрас вершины в зеленый цвет отличает её от других и означает, она больше не будет использовать в расчётах путей. Далее последовательно выполняются следующие действия, пока есть нефиксированные вершины.

1. Алгоритм рассматривает всех нефиксированных соседей от текущей вершины и шагом является одно такое рассмотрение, при котором вес после изменения стал меньше предыдущего. Цвет рассматриваемой вершины меняется на желтый. На следующем шаге перекрашиваем прошлую вершину из жёлтого в фиолетовый. И так до тех пор, пока не будут рассмотрены все такие соседи.

2. Алгоритм выбирает из всех нефиксированных вершин, вершину с минимальным весом. Если таких несколько, то берётся любая из вершин. Шагом является фиксация вершины и последующая перекраска в зелёный цвет.

У пользователя есть возможности взаимодействия с шагами алгоритма в соответствии с кнопками, описанными выше. Таким образом, пользователь сможет: посмотреть проигрывание алгоритма с задержкой в 5 секунд, сам переходить на шаг вперёд или назад в алгоритме или вовсе, минуя все шаги, получить результат работы алгоритма Дейкстры.

Если пользователь выбирает проигрывание алгоритма, то кнопка проигрыша меняется на кнопку приостановки алгоритма. При нажатии на данную кнопку алгоритм остановится на текущем шаге, после остановки пользователь сможет продолжить алгоритм с текущего шага в удобной ему форме.

Информационная панель используется для вывода пояснений при работе алгоритма. Вывод данных осуществляется в виде текста.

1.1.2. Требования к вводу исходных данных

Исходные данные вводятся либо за счет графического интерфейса, либо за счет парсинга данных из файла.

При задании графа через файл, данные должны быть помещены в файл с расширением .txt и записаны по некоторым правилам.

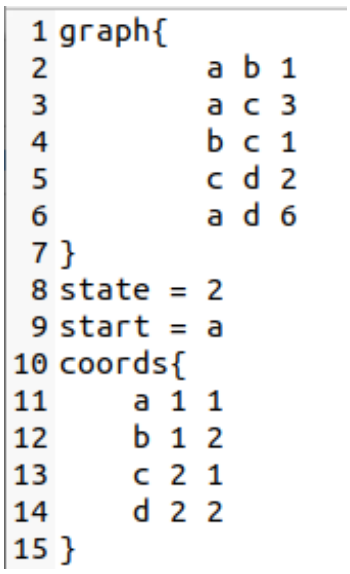


Рисунок 2. Пример задания входных данных

Для задания графа используется ключевое слово «graph», далее необходимо на той же строке открыть блок информации символом «{», таким образом программа поймет что ниже располагаются ребра графа и их веса, разделённые пробельными символами как на картинке. Синтаксис задания ребра: запись «a c 3», говорит программе о том, что между вершинами «a» и «c» существует ребро весом 3, таким образом можно задать весь желаемый граф. После перечисления всех рёбер, на следующей строке необходимо поставить «}», сигнализирующую об окончании блока информации. Задание графа в указанном файле является обязательным для считывания программой.

Схожим образом задается обязательный блок информации «coords» сообщающей программе о примерном, желаемом расположении вершин графа на холсте программы. Запись «c 2 1», сообщает программе о том, что вершина «c» имеет X координату равную 2 и Y координату равную 1. Таким образом можно задать координаты каждой вершины. Для создания блока информации также необходимо отделить его фигурными скобками. Для каждой вершины необходимо задать координаты.

Для работы программы ключевые слова «state» и «start» являются необязательными задаваемыми. Ключевое слово «state» отвечает за номер шага алгоритма. Если указать его, то алгоритм откроется на шаге i-м после знака равно в записи «state = i», если все остальные данные заданы корректно.

Ключевое слово «start», отвечает за вершину старта. И запись «start = a» говорит программе о том, что вершина «a» является вершиной старта.

Указанный порядок ключевых слов не является обязательным, а лишь служит демонстрационным примером задания корректных данных. Для корректного считывания описанной информации блоки не должны пересекаться.

В случае, если граф задается графически, вышеописанная панель инструментов с добавлением и удалением элементов графа позволит корректно создать граф.

1.1.3. Требования к структуре программы

Явное разделение отрисовки и бизнес-логики. Создание отдельных пакетных модулей под логику и отрисовку.

Для очевидного разделения отрисовки и бизнес логики в проекте используется паттерн MVC: созданы классы, представляющие собой отдельно логику(Logic), отрисовку(UI) и контроллер(Tools).

Схема взаимодействия классов представлена UML-диаграммой.

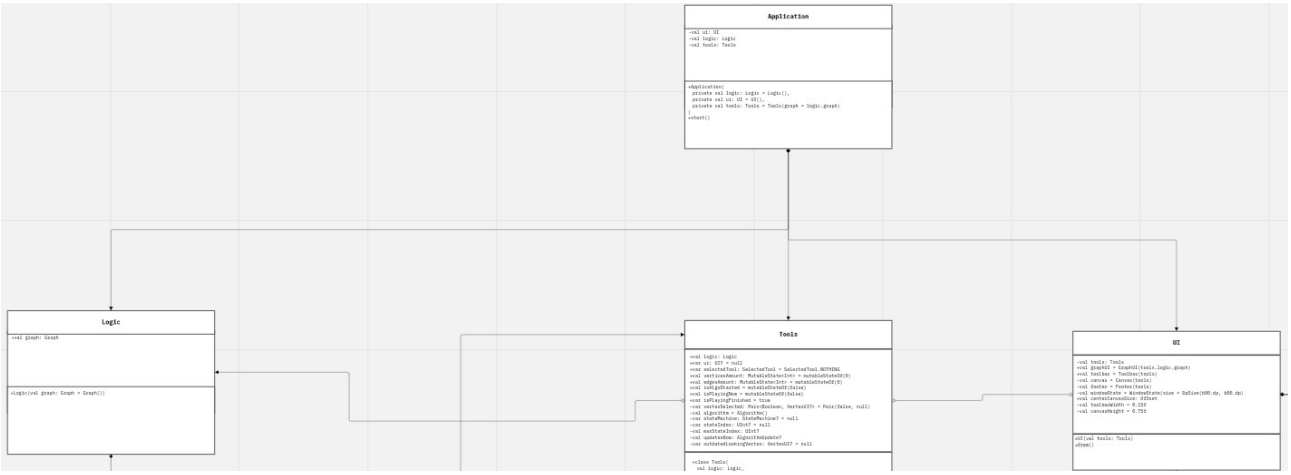


Рисунок 3. UML

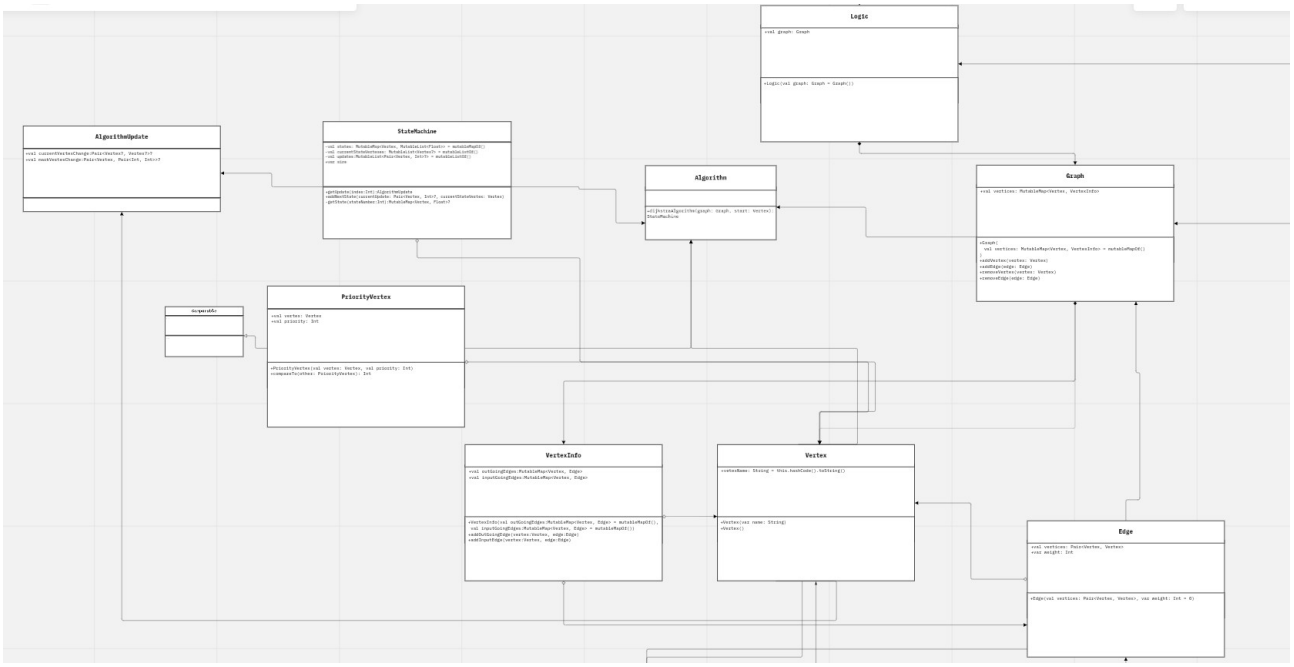


Рисунок 4. UML

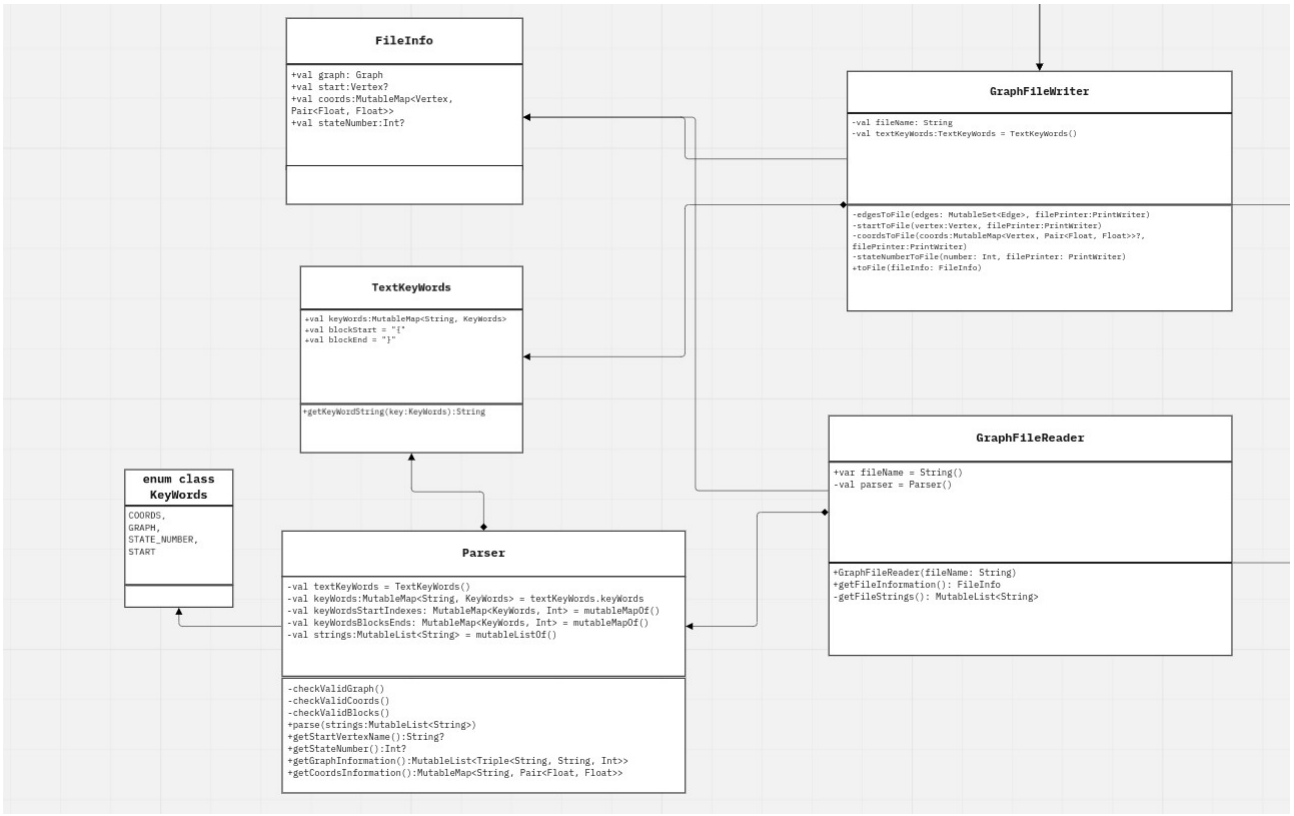


Рисунок 5. UML

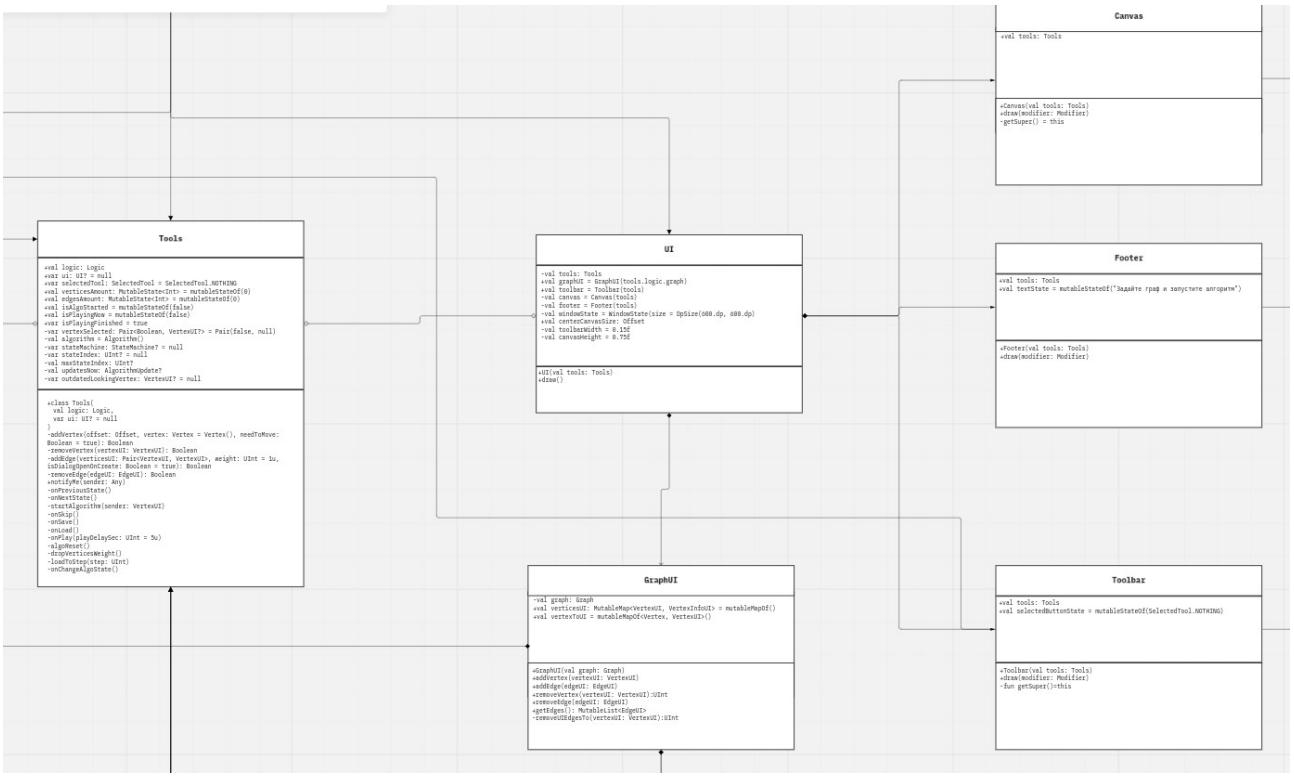


Рисунок 6. UML

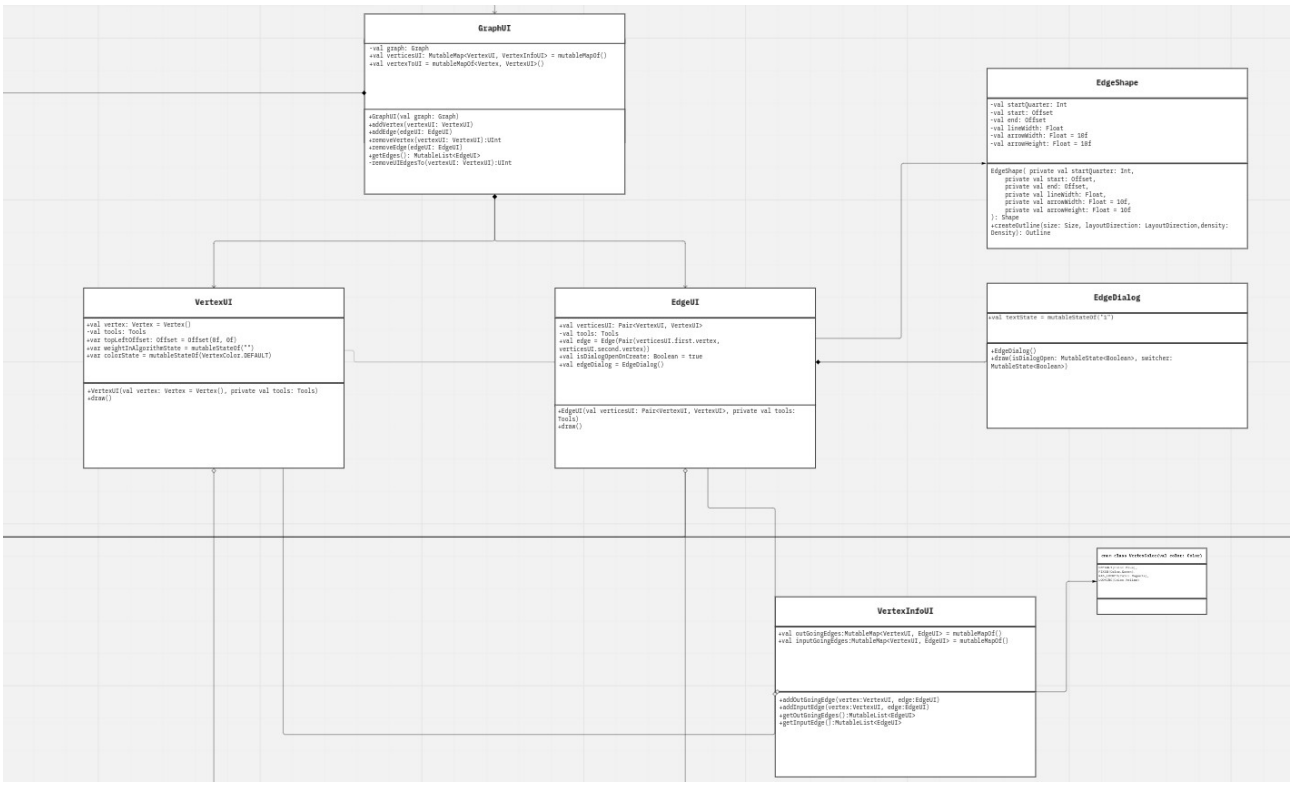


Рисунок 7. UML

1.1.4. Требования к коду

Отсутствие повтора кода, спагетти-кода, а также использование ООП парадигмы.

1.1.5. Требования к языку

Написание программы на ЯП Kotlin с графической отрисовкой Compose Multiplatform от JetBrains и логгированием SLF4J.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Дата	Содержание задания	Статус(выполнено или нет)
01.07.22	Попытка сдача спецификации программы.	-
02.07.22	Финальная сдача спецификации программы.	-
03.07.22	Разработка каркаса проекта. Добавление панели инструментов и холста. Появление возможности отрисовки компонент графа.	+
04.07.22	Согласование спецификации. Сдача прототипа.	-
05.07.22	Добавление возможности ввода веса ребер. Реализация алгоритма Дейкстры.	+-
07.07.22	Согласование спецификации. Показ 1-ой итерации.	+
07.07.22	Отрисовка веса до вершины на текущем шаге. Добавление машины состояний.	+
08.07.22	Показ 2-ой итерации.	+
09.07.22 — 11.07.22	Добавление возможности ввода данных с файла. Сохранение выходного состояния в файл. Тестирование, дебаггинг. Написание отчета.	+
12.07.22	Сдача финальной версии.	

2.2. Распределение ролей в бригаде

Никитин Д.Э. - проектировщик, разработчик графического интерфейса.

Нагибин И.С. - разработчик логики, тестировщик.

Жиглов Д.С. - тестировщик.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Программа

Класс Application

Application — класс, содержащий 3 основных объекта: модель(Logic), вид(UI), контроллер(Tools).

Логика

Класс Logic

Класс Logic архитектурно представляет логику программы. Хранит в себе граф. Отвечает за изменение логического уровня программы.

Класс Graph

На логическом уровне представляет собой граф, которому можно добавлять отдельные вершины и ребра между ними. Также можно удобно брать информацию о графе из его методов. Сам граф хранится в виде хэш-таблицы, в которой каждой вершине(Vertex) соответствует информация о входящих и исходящих ребрах(VertexInfo).

Класс Vertex

Представляет собой вершину графа(Graph). Класс хранит в себе имя вершины, если имя не задается, то в роли имени выступает хэш-код объекта.

Класс Edge

Представляет собой ребро между двумя вершинами. Хранит в себе обе вершины в виде пары и вес ребра между ними. Вес ребра изменяем.

Класс VertexInfo

Отвечает за информацию об одной вершине графа. Хранит в себе соответствия вершина — ребро для входящих и исходящих рёбер. В качестве вершины задается откуда или куда ведёт ребро. То есть для исходящих ребёр будет указана конечная вершина ребра, а для входящих наоборот.

Класс Algorithm

Отвечает за вычисление всех шагов алгоритма Дейкстры и формирует машину состояний для удобной пошаговой отрисовки. Сам алгоритм Дейкстры представлен в виде метода и возвращает машину состояний(StateMachine).

Класс PriorityVertex

Хранит в себе вершину и её указанный приоритет. Наследуясь от интерфейса Comparable, таким образом реализуется перегрузка метода сравнения, в котором сравниваются приоритеты вершин. Используется в классе Algorihm в очереди с приоритетом

Класс AlgorithmUpdate

Информационный(data) класс. Является представлением обновления с предыдущего шага алгоритма Дейкстры. Хранит в себе обновление обрабатываемой вершины в виде пары вида «было, стало» и вершину, метка которой поменялся в текущем обновлении алгоритма Дейкстры, тоже в соответствии пары вида «было, стало». Экземпляр класса возвращается в методе при взятии обновления в машине состояний(StateMachine).

Класс StateMachine

Представляет собой машину состояний алгоритма Дейкстры. В классе хранится весь список вершин графа и список состояний на каждом шаге каждой из них в виде соответствия. В отдельном поле хранится текущая вершина на

каждом шаге. Также существует поле в котором хранятся все обновления меток вершин в графе. Хранится также количество шагов алгоритма Дейкстры. Храня все вышеперечисленные поля можно по одному лишь обновлению меток вершин формировать информацию о метках вершин всего графа на каждом шаге.

В алгоритме Дейкстры на каждом шаге машине сообщается об обновлении какой-либо метки графа. После окончания алгоритма будет сформирована машина состояний графа, способна пошагово передавать полученные обновления графа.

Класс FileInfo

Информационный(data) класс. Используется для передачи информации из файла и в файл. Хранит в себе граф, стартовую вершину, координаты вершин графа и номер, на котором воспроизведение алгоритма Дейкстры должно начаться при запуске.

Класс KeyWords

Класс перечисления(enum), хранящий ключевые слова в виде перечисления.

Класс TextKeyWords

Хранит в себе текстовое представление ключевых слов их соответствие из перечисления KeyWords. Также хранит символы отвечающие за начало и конец информационного блока.

Класс Parser

Сущность ответственная за проверку текстовых данных, введённых в файл, на корректность. Работает только с строками, полученными из файла. После проверки корректности ведённых данных может вернуть информацию о

загруженных в файл объектах. В случае некорректных данных сообщит об этом в виде ошибки экземпляра класса IOException.

Хранит в себе экземпляры классов TextKeyWords, KeyWords, а также начала и концы блоков информации. Все строки переданные для анализа также хранятся в качестве поля класса.

Класс GraphFileReader

Класс отвечает за считывание данных из файла. Считывает строки из файла, а проверку на корректность делегирует классу Parser.

Полученную из Parser'а информацию проверяет на корректность и возвращает в виде структуры FileInfo. В случае некорректных данных сообщит об этом в виде ошибки экземпляра класса IOException.

Класс GraphFileWriter

Сущность ответственная за запись данных FileInfo в файл. Если возникнут проблемы с записью файл сообщит об этом в виде ошибки экземпляра класса IOException.

Отрисовка

Класс UI

UI — класс отрисовки. Имеет объект класса WindowState, представляющий собой окно. Хранит в себе основные объекты отрисовки: Toolbar, Canvas, Footer, а также graphUI. Принимает ссылку на объект Tools, за счёт чего можно передавать изменения в отрисовке в Tools для реагирования на изменения.

Панели:

Класс Canvas

Canvas — класс, реализующий холст, на котором рисуется граф. Реализован в виде layout типа Box, за счёт чего в любое место на экране можно добавить объект.

Класс Footer

Footer — класс, представляющий собой панель, на которой выводится текст с шагами алгоритма.

Класс Toolbar

Toolbar — панель инструментов, содержащая кнопки, описанные в спецификации.

Графические объекты

Класс GraphUI

GraphUI — класс, представляющий собой графическую оболочку над Graph. Хранит Graph, словарь пар вида <VertexUI, VertexInfoUI>, а также словарь соответствий Vertex в VertexUI. Принимает ссылку на объект Tools, за счёт чего можно передавать изменения в отрисовке в Tools для реагирования на изменения.

Класс VertexUI

VertexUI — оболочка над Vertex. Хранит координаты в виде Offset. Имеет вес работы алгоритма в виде String, а также цвет в виде объекта из перечисления VertexColor. Принимает ссылку на объект Tools, за счёт чего можно передавать изменения в отрисовке в Tools для реагирования на изменения.

Класс EdgeUI

EdgeUI — оболочка над Edge. Хранит пару VertexUI, как ребро. Хранит объект edgeDialog — окна, позволяющего изменить вес ребра, через диалоговое

окно, а также состояние диалога(открыт или закрыт). Принимает ссылку на объект Tools, за счёт чего можно передавать изменения в отрисовке в Tools для реагирования на изменения.

Класс EdgeShape

EdgeShape — класс, представляющий форму ребра. Хранит параметры:

- четверь, в которой находится стартовая вершина
- координаты старта и конца, на окружности,
- ширина линии
- размеры стрелки

Класс EdgeDialog

EdgeDialog — диалоговое окно, позволяющее задать вес ребра.

Класс VertexInfoUI

VertexInfoUI — дополнительный класс, хранящий входящие и выходящие ребра EdgeUI в GraphUI для каждой вершины. Enum class VertexColor позволяет задать цвет вершины.

Контроллер

Класс Tools

- Tools — класс контроллера в MVC. Содержит следующие поля:
- logic — ссылка на Logic
- ui — ссылка на UI
- selectedTool — текущий выбранный инструмент
- verticesAmount — количество вершин
- edgesAmount — количество ребер
- isAlgoStarted — запущен ли алгоритм
- isPlayingNow — проигрывается ли алгорит сейчас
- isPlayingFinished — закончилось ли проигрывание

- vertexSelected — пара <выбранна ли вершина для создания ребра, выбранная вершина>
- algorithm — класс Algorithm
- stateMachine - StateMachine
- stateIndex — номер шага в алгоритме
- maxStateIndex — номер максимального шага
- updatesNow — поле, позволяющее получить обновления на текущем шаге
- outdatedLookingVertex - для удобной отрисовки вершины из VertexColor.LOOKING в VertexColor.WAS_LOOKED

Реализован Tools, как медиатор, что позволило сделать из него контроллер.

3.2. Основные методы

Программа

Класс Application

fun start() - запускает приложение(графическую отрисовку)

Логика

Класс Graph

1. addVertex(vertex: Vertex)

   Добавляет в граф вершину vertex.
2. addEdge(edge: Edge)

   В случае если вершины ребра лежат в графе и вес ребра неотрицательный добавляет в граф ребро edge.
3. removeEdgesTo(vertex:Vertex)

   Приватный метод удаляющий ребра к указанной вершине vertex.
4. removeVertex(vertex: Vertex)

   Если вершина vertex присутствует в графе вызывает метод removeEdgesTo, удаляющий ребра ведущие к данной вершине и из неё, после вершина удаляется из графа.

5. removeEdge (edge: Edge)

   Если ребро edge находится в графе, удаляет его.

6. getDestinations(vertex:Vertex)

   Если вершина vertex в графе, то возвращает все исходящие ребра в виде набора.

7. GetVertices()

   Возвращает все вершины графа в виде набора.

8. GetEdges()

   Возвращает все ребра графа в виде набора.

Класс VertexInfo

1. addOutGoingEdge(vertex:Vertex, edge:Edge)

   Добавляет в соответствие исходящих рёбер ребро edge, ведущее к вершине vertex.

2. addInputEdge(vertex:Vertex, edge:Edge)

   Добавляет в соответствие входящих рёбер ребро edge, исходящее от вершины vertex.

Класс GraphFileReader

1. getFileStrings()

   Приватный метод. Считает из указанного файла, который было передан в конструктор класса строки файла, если это возможно, в ином случае сообщит об этом в виде ошибки экземпляра класса IOException..

2. getFileInformation()

   Возвращает FileInfo в соответствии с заданными в файле данными. Если файл составлен некорректно сообщит об этом в виде ошибки экземпляра класса IOException.

Класс GraphFileWriter

1. edgesToFile(edges: MutableSet<Edge>, filePrinter:PrintWriter)

   Приватный метод. Запишет в файловый поток filePrinter ребра графа, преданные в качестве параметра edges.

2. startToFile(vertex:Vertex, filePrinter:PrintWriter)

   Приватный метод. Запишет вершину старта vertex в поток filePrinter.
3. coordsToFile(coords:MutableMap<Vertex, Pair<Float, Float>>?, filePrinter:PrintWriter)

   Приватный метод. Схожим образом запишет координаты coords вершин графа в файловый поток filePrinter.
4. stateNumberToFile (number: Int, filePrinter: PrintWriter)

   Приватный метод. Запишет шаг алгоритма Дейкстры в файловый поток filePrinter.
5. toFile(fileInfo: FileInfo)

   Вызовет все вышеперечисленные методы для записи данных fileInfo со всеми необходимыми проверками.

   Класс Parser
1. parse(strings:MutableList<String>)

   Анализирует переданные строки файла strings, обнаруживая и записывая начала блоков информации. Если блоков отвечающих за граф или за координаты его вершин не обнаружено будет выведена ошибка. Также метод вызывает приватный методы проверка всех возможных составляющих графа.
2. CheckValidGraph()

   Приватный метод. Анализирует ребра графа и передаваемые в качестве начала и конца ребра вершины на корректность.
3. CheckValidCoords()

   Приватный метод. Анализирует координаты вершин графа на их корректность.
4. CheckValidBlocks()

   Приватный метод. Анализирует не пересекается ли блоки информации между собой.
5. GetGraphInformation()

Возвращает проанализированные данные в виде списка экземпляров класса Triple<String, String, Int>, в которых хранится вершина начала ребра, вершина конца ребра и значение веса самого ребра.

6. GetCoordsInformation()

   Возвращает проанализированные данные в виде хэш-таблицы, у которой в качестве ключа выступает имя вершины типа String, а в качестве значения пара с координатами вершины.

7. GetStartVertexName()

   Если вершина старта была задана, то вернёт её название типа String, иначе null.

8. GetStateNumber()

   Если номер шага алгоритма задан, то вернет число типа Int, иначе null.

Класс StateMachine

1. getUpdate(index:Int)

   Возвращает обновление(AlgorithmUpdate) с номером index, если idnex < size, иначе вернется AlgorithmUpdate(null, null).

2. addNextState(currentUpdate: Pair<Vertex, Int>?, currentStateVertex: Vertex)

   Добавляет в машину состояний следующее состояние. В качестве первого аргумента передается пара вершина и новая метка указанной вершины, далее передается текущая вершина от которой считалась метка.

3. getState(stateNumber:Int)

   Приватный метод, который принимает на вход Int stateNumber — номер шага алгоритма, если шаг с таким номером был совершен вернёт MutableMap<Vertex, Float>, которая хранит в качестве ключа вершину графа, а в качестве значения метку вершины на указаном шаге.

Класс TextKeyWords

1. getKeyWordString(key:KeyWords)

   Возвращает строку(ключевое слово), которая соответствует переданному экземпляру класса KeyWords key. Иначе вернёт "NOT FOUND KEY-WORD STRING".

Отрисовка

Класс UI

1.fun draw() - метод, рисующий весь UI.

Панели:

Класс Canvas

1. fun draw(modifier: Modifier) — метод, рисующий Canvas

Класс Footer:

1. fun draw(modifier: Modifier) — метод, рисующий Footer

Класс Toolbar

1. fun draw(modifier: Modifier) — метод, рисующий Toolbar

Графические объекты

Класс GraphUI

1. fun addVertex(vertexUI: VertexUI) — добавление VertexUI в GraphUI
2. fun addEdge(edgeUI: EdgeUI) — добавление EdgeUI в GraphUI
3. private fun removeUIEdgesTo(vertexUI: VertexUI):Uint — удалить все ребра, связанные с данным vertexUI и вернуть их количество
4. fun removeVertex(vertexUI: VertexUI):Uint — удалить vertexUI и удалить все связанные с ним ребра. Вернуть количество удаленных ребер.
5. fun removeEdge(edgeUI: EdgeUI) — удаление ребра
6. fun getEdges():MutableList<EdgeUI> - получить все ребра

Класс VertexUI

1. fun draw() - отрисовка вершины

Класс EdgeUI

1. fun draw() - отрисовка ребра

Класс EdgeShape

1. override fun createOutline(
   size: Size,
   layoutDirection: LayoutDirection,
   density: Density
   ): Outline — метод, позволяющий построить замкнутый контур, задающий форму ребра.

Класс EdgeDialog

1. fun draw(isDialogOpen: MutableState<Boolean>, switcher: MutableState<Boolean>) - метод, рисующий диалоговое окно

Класс VertexInfoUI

1. fun addOutGoingEdge(vertex:VertexUI, edge:EdgeUI) — добавить исходящее ребро
2. fun addInputEdge(vertex:VertexUI, edge:EdgeUI) - добавить входящее ребро
3. fun getOutGoingEdges():MutableList<EdgeUI> - получить исходящие ребра
4. fun getInputEdge():MutableList<EdgeUI> - получить входящие ребра

Контроллер

Класс Tools

1. fun notifyMe(sender: Any) — метод для медиатора, позволяющий принимать обновления.
2. private fun addVertex(offset: Offset, vertex: Vertex = Vertex(), needToMove: Boolean = true): Boolean — добавление вершины по заданным координатам в graphUI
3. fun removeVertex(vertexUI: VertexUI): Boolean - удаление вершины в graphUI

4. private fun addEdge(verticesUI: Pair<VertexUI, VertexUI>, weight: UInt = 1u, isDialogOpenOnCreate: Boolean = true): Boolean - добавление ребра в graphUI
5. private fun removeEdge(edgeUI: EdgeUI): Boolean - удаление ребра в graphUI
6. private fun onPreviousState() - шаг назад по машине состояний
7. private fun onNextState() - шаг вперед по машине состояний
8. private fun startAlgorithm(sender: VertexUI) - запуск алгоритма от выбранной вершниы
9. private fun onSkip() - переход к последнему шагу в машине состояний
10. private fun onSave() - сохранить граф и шаг алгоритма
11. private fun onLoad() - загрузить граф и шаг алгоритма
12. private fun onPlay(playDelaySec: UInt = 5u) — воспроизведение всех шагов алгоритма с задержкой в 5 секунд.
13. private fun algoReset() - сброс данных, касающихся алгоритма
14. private fun dropVerticesWeight() - сброс весов у всех вершин
15. private fun loadToStep(step: Uint) — переход до шага step
16. private fun onChangeAlgoState() - при изменении состояния запуска алгоритма.

4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса

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

- Режима изменения графа
- Режима воспроизведения алгоритма Дейкстры
- Инструментов сохранения и загрузки

Режим изменения графа

1) Тестирование добавления новой вершины графа.

Ожидание: нажатие на клавишу “+V” и появления на пустом холсте новой вершины.

Что имеется:

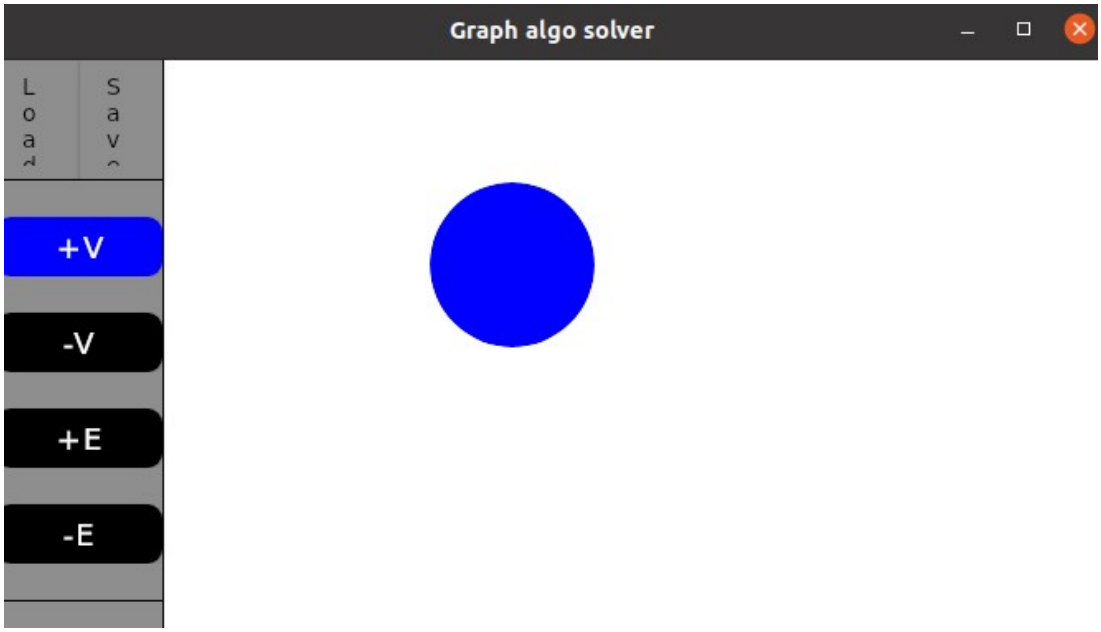


Рисунок 8 – Добавление вершины

При нажатии на клавишу добавления графа сама клавиша меняет цвет на синий, и после нажатия на любую область на холсте появятся вершина, покрашенная в синий цвет.

2) Удаления вершины из графа:

Ожидание: при нажатии на клавишу “-V” и после нажатие на вершину, которая должна пропасть, удаление этой вершины.

Что имеется:

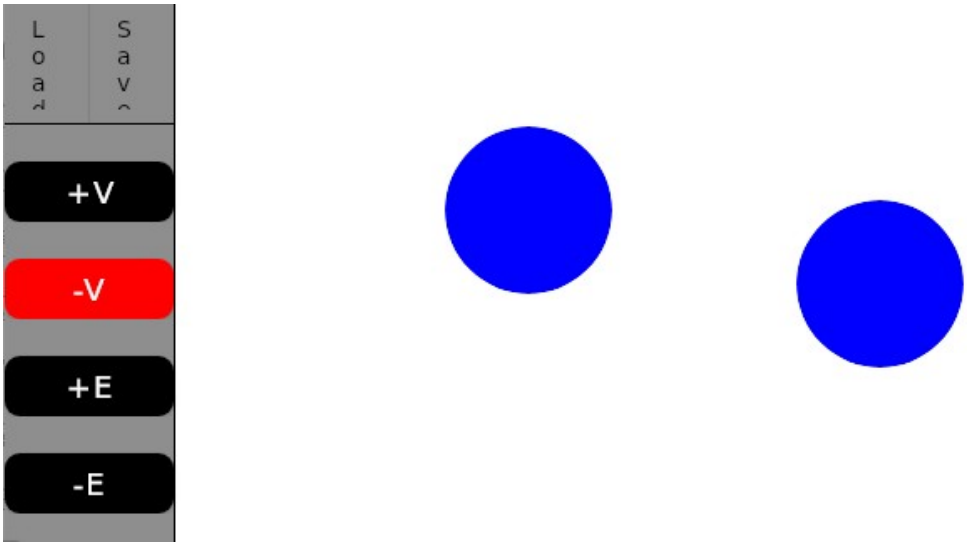


Рисунок 9 – Добавление вершины

При нажатии на клавишу удаления вершины сама клавиша окрашивается в красный цвет, и после, при нажатии на любую вершину, она пропадает из графа.

3) Добавление ребер между вершинами.

Ожидание: при нажатии на клавишу “+F” и соединении двух вершин должно появиться ребро, так же должна быть возможность задать вес этого ребра.

Что имеется:

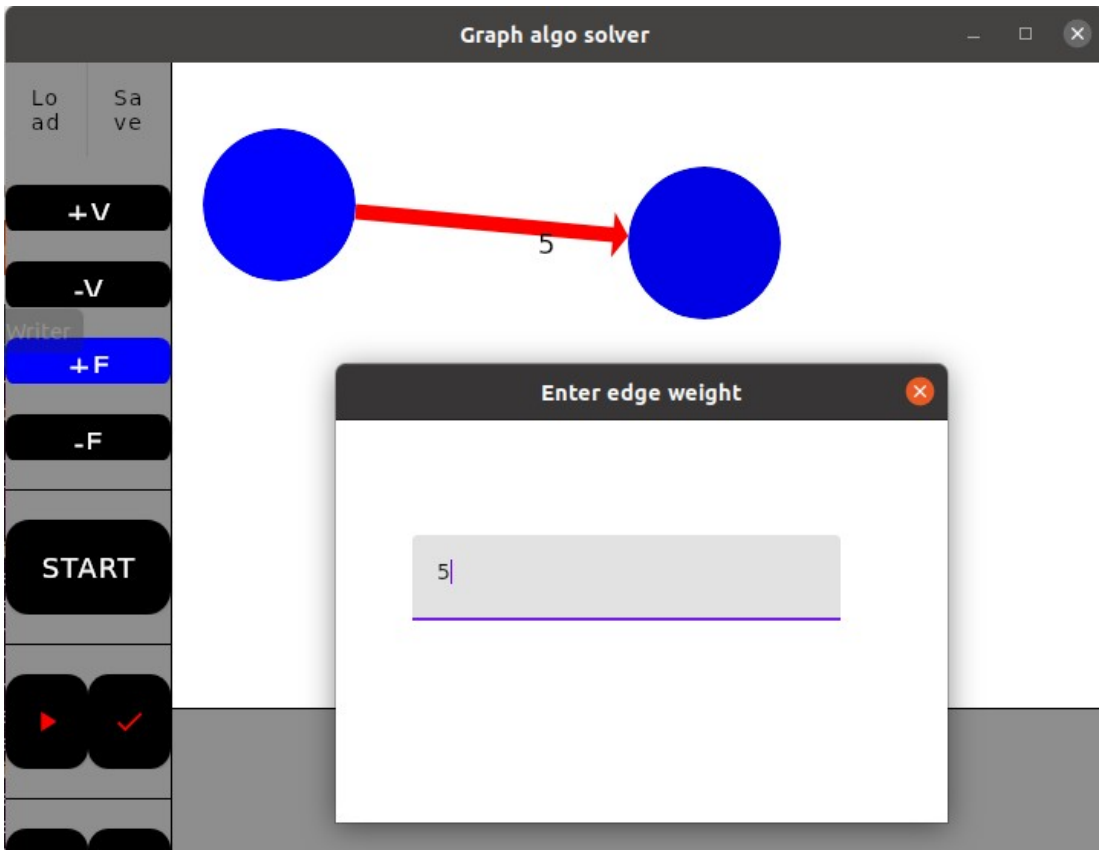


Рисунок 10 – Добавление вершины

При нажатии на клавишу добавления ребра и после этого нажатия на 2 вершины, которые нужно соединить, между этими вершинами появляется ребро и всплывает окошко, где необходимо указать вес данного ребра.

4) Удаление ребра между вершинами.

Ожидание: при нажатии на клавишу “-F” и указании, между какими 2 вершинами необходимо удалить связь, эта связь удаляется.

Что имеется:

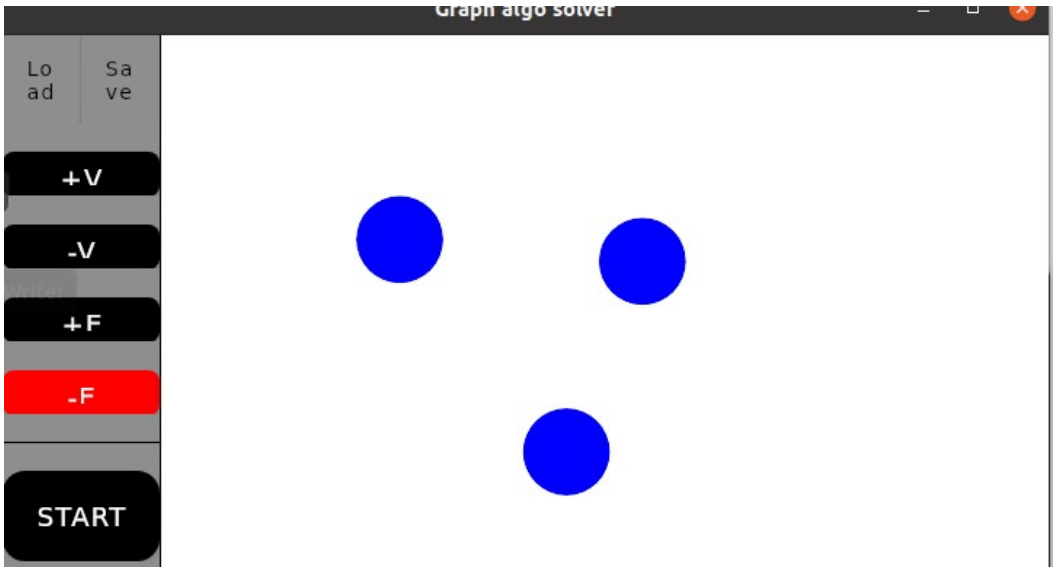


Рисунок 11 – Добавление вершины

При нажатии на клавишу удаления ребра и после нажатия на само ребро, ребро благополучно удаляется.

Режима воспроизведения алгоритма Дейкстры

Ожидание: должна быть возможность задачи вершины старта, после должен быть показ ход работы алгоритма, изменение стоимости пути до вершин.

Что имеется:

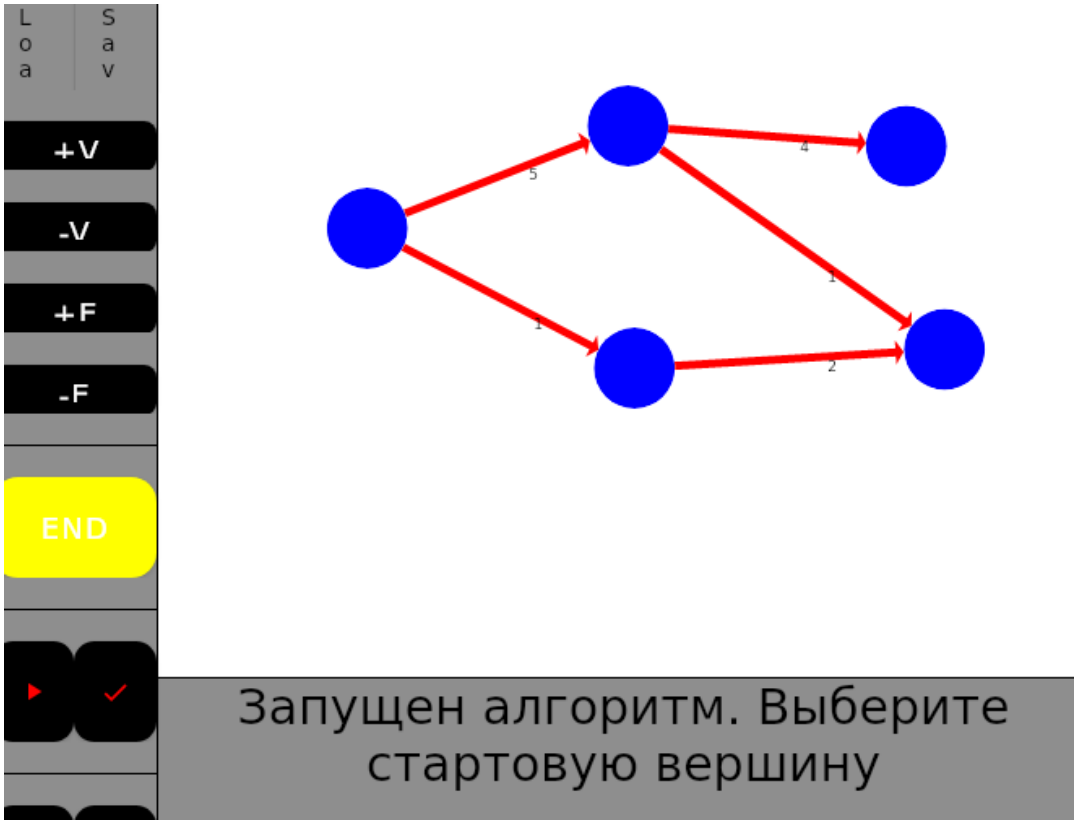
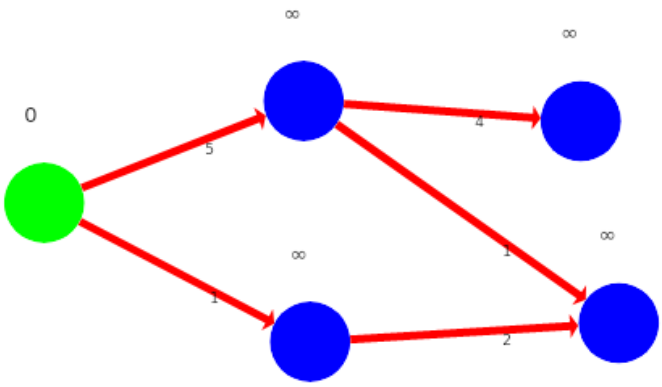


Рисунок 12 – Добавление вершины

1) Сначала запрашивается указание стартовой вершины.



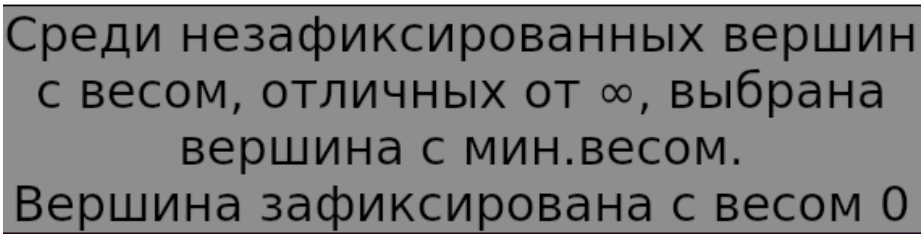


Рисунок 13 – Добавление вершины

2) При задании графа и нажатии на кнопку старт необходимо выбрать стартовую вершину, после ее выбора стоимость до этой вершины будет равняться 0, а до остальных стоимость будет равна бесконечности.

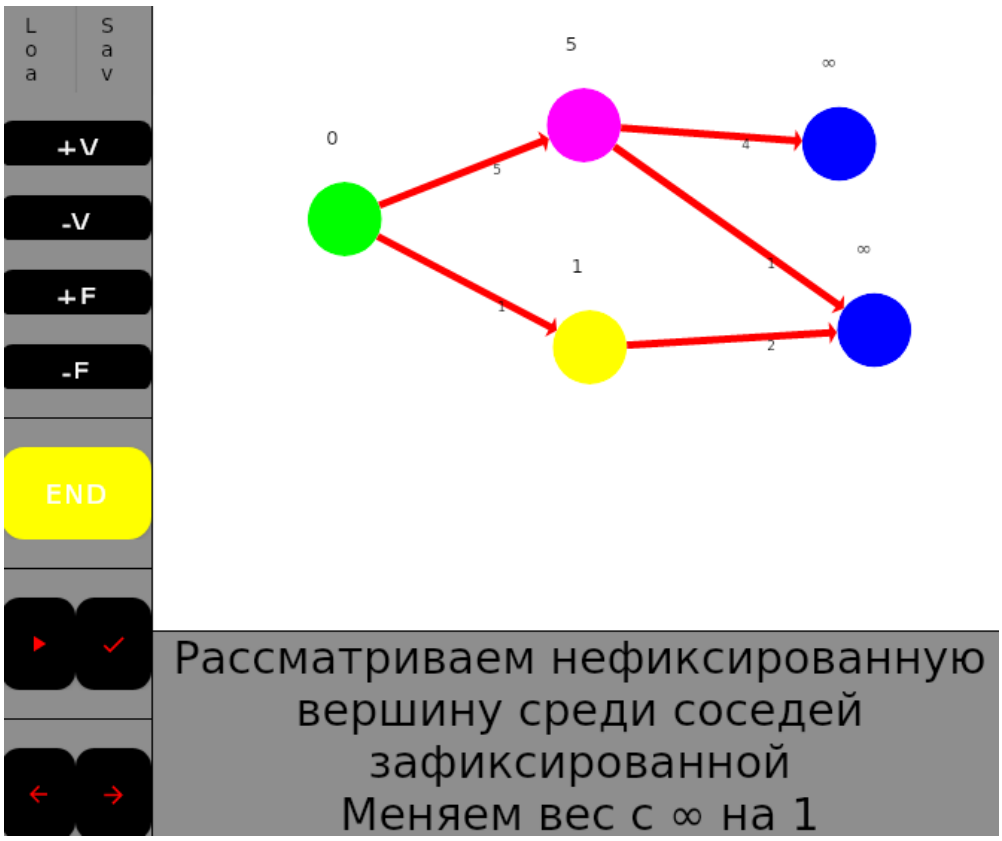


Рисунок 14 – Цвета вершин во время работы алгоритма и изменения весов

3) Далее, при ходе работы алгоритма, вершины, пути из которых еще рассматриваются, окрашиваются в пурпурный цвет, вершины, из которых ищется путь в данный момент, окрашиваются в желтый, а вершины, которые уже были рассмотрены и путь до них является конечным, окрашиваются в зеленый цвет.

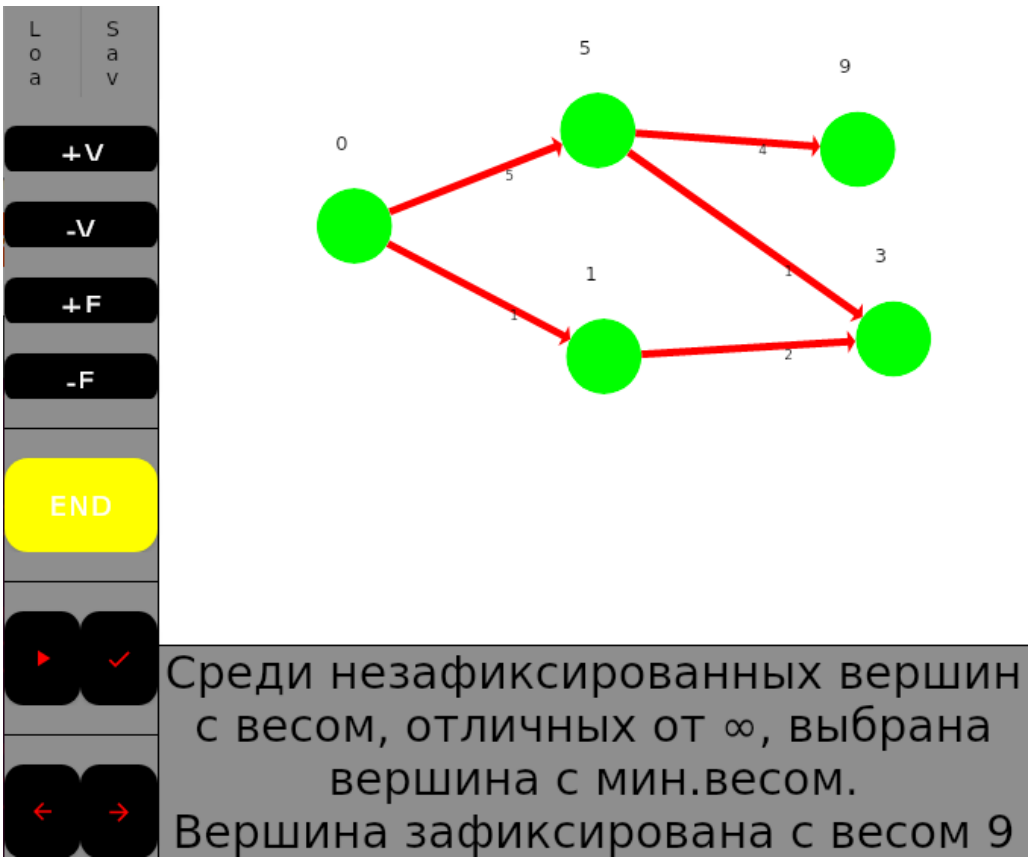


Рисунок 15 – Конец работы агоритма

4) В конце работы алгоритма все вершины оказываются окрашеными в зеленый цвет, и над всеми вершинами есть цифра, которая указывает стоимость пути до этой вершины.

Так же есть возможность пропуска пошаговой работы алгоритма при нажатии на кнопку галочка.

Рисунок 16 – Кнопка для пропуска шагов работы алгоритма

5) Имеется и показ пошаговой работы с задержкой в 5 секунд при нажатии на кнопку “play”.

Рисунок 17 – кнопка для показа пошаговой работы алгоритма с задержкой в 5 секунд

Инструментов сохранения и загрузки

Ожидание: Должна быть возможность сохранения графа при работе с ним, а после возможность загрузки его исходного состояния.

Что имеется:

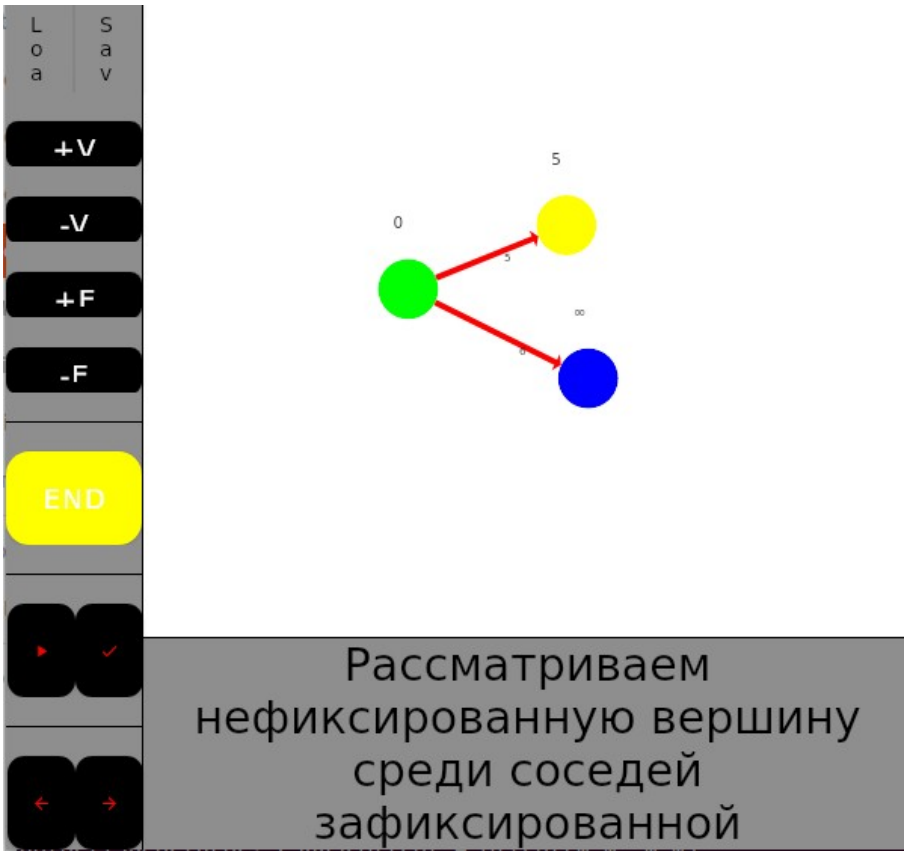


Рисунок 18 – Загруженный граф

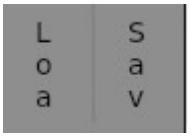


Рисунок 19 – кнопки для сохранения и загрузки графа

В текущей итерации алгоритма нажимается кнопка save и граф и шаг алгоритма сохраняется, после можно перезапустить jar-файл, нажать на кнопку load и загрузится граф с исходного состояния.

4.2. Тестирования кода графа

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 двумя-тремя тестами: тестом на нормальную работу, тестом на нормальную работу в нетривиальных обстоятельствах или на получения ошибки при других обстоятельствах.

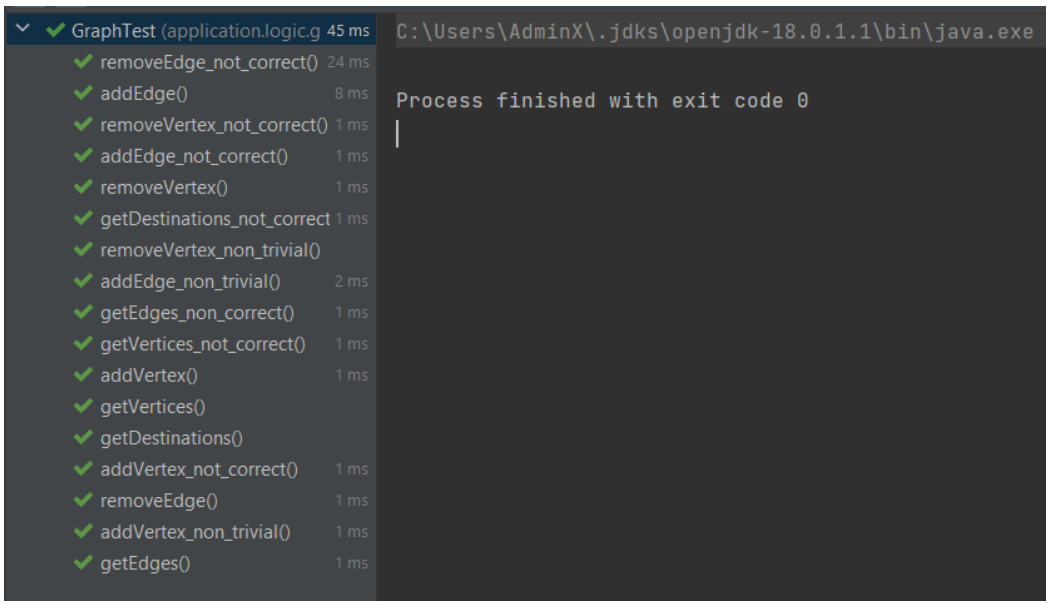


Рисунок 20 – Тесты для графа

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 графе, а также ситуации, когда путь не существует, либо же когда существуют нескольк путей.

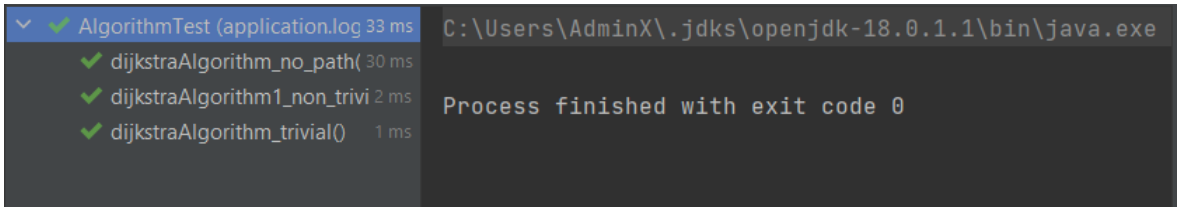


Рисунок 21 – Тестирование алгоритма на графе

4.4. Тестирование кода парсера

Для тестирования кода парсера были подобраны различные входные данные: с заданными координатами, без заданных координат, различное местоположение блоков заданных данных. Во всех этих случаях парсер отрабатывал корректно и правильно считывал информацию.

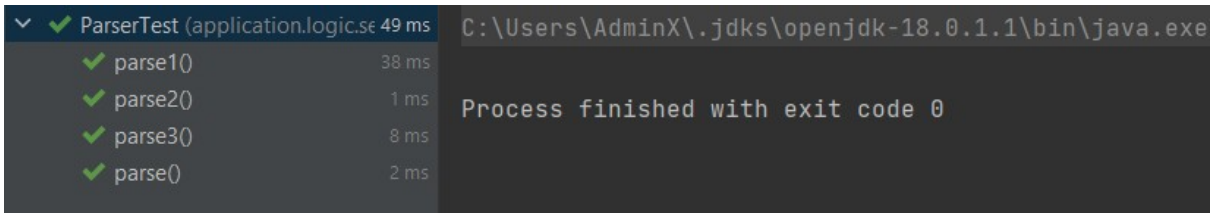


Рисунок 22 –Тестирование парсера

ЗАКЛЮЧЕНИЕ

Подводя итоги проделанной работы, выполнение всех поставленных задач было выполнено в короткие сроки с минимальным количеством проблем. Некоторые концепции и способы реализации были подкорректированы в процессе разработки, опираясь на реальность их реализации за поставленное время. Тем не менее, в конечном итоге получился качественный продукт.

Конечно, в программе могло остаться некоторое количество багов, которые не были обнаружены при тестировании. На их поиск и исправление требуется дополнительное время, которое нельзя уместить в заложенные сроки практики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Руководство по языку Kotlin // KotlinLang. URL: https://kotlinlang.ru/ .

2. Примеры использования Jetpack Compose для Kotlin Multiplatform // Github. URL: https://github.com/jetbrains/compose-jb (дата обращения: 02.07.2022)

3. Уроки по работе с Jetpack Compose // Jetpack Compose Tutorial. URL: https://www.jetpackcompose.net/jetpack-compose-introduction (дата обращения: 04.07.2022).

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ПРИЛОЖЕНИЯ

kotlin/application/controller/tools.kt

```
package application.controller

import GraphFileWriter
import androidx.compose.runtime.mutableStateOf
import androidx.compose.ui.ExperimentalComposeUiApi
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.input.key.*
import application.logic.Logic
import application.logic.serialization.AlgorithmUpdate
import application.logic.serialization.FileInfo
import application.ui.UI
import application.ui.objects.EdgeUI
import application.ui.objects.VERTEX_SIZE
import application.ui.objects.VertexColor
import application.ui.objects.VertexUI
import application.ui.window.Canvas
import application.ui.window.Toolbar
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.cancel
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import logger
import logic.*

enum class SelectedTool {
    NOTHING, ADD_VERTEX, REMOVE_VERTEX, ADD_EDGE, REMOVE_EDGE
}

class Tools(
    val logic: Logic,
    var ui: UI? = null
) {
    var selectedTool: SelectedTool = SelectedTool.NOTHING
    val verticesAmount = mutableStateOf(0)
    val edgesAmount = mutableStateOf(0)
```

```
    val isAlgoStarted = mutableStateOf(false)
    val isPlayingNow = mutableStateOf(false)
    var isPlayingFinished = true

    private var vertexSelected = Pair<Boolean, VertexUI?>(false, null)

    private val algorithm = Algorithm()

    private var stateMachine: StateMachine? = null
    private var stateIndex: UInt? = null

    private val maxStateIndex: UInt?
        get() {
            return stateMachine?.size?.toUInt()?.minus(1u)
        }
    private val updatesNow: AlgorithmUpdate?
        get() {
            return stateIndex?.toInt()?.let { stateMachine?.getUpdate(it)
}
        }

    private var outdatedLookingVertex: VertexUI? = null

    @OptIn(ExperimentalComposeUiApi::class)
    fun notifyMe(sender: Any) {
        when(sender) {
            is Pair<*, *> -> {
                if (sender.first is UI && sender.second is KeyEvent) {
                    val it = sender.second as KeyEvent
                    if (it.key == Key.Tab && it.type ==
KeyEventType.KeyDown) {
                        ui?.toolbar?.selectedButtonState?.value =
SelectedTool.NOTHING
                    }
                }
                else if (sender.first is Canvas && sender.second is
Offset) {
                    when(selectedTool) {
                        SelectedTool.ADD_VERTEX -> {
```

```
                            logger.info("selectedTool = $selectedTool, so
addVertex")
                            addVertex(sender.second as Offset)
                        }
                    }
                }
                else if (sender.first is Toolbar && sender.second is
String) {
                    when (sender.second) {
                        "previousState" -> {
                            onPreviousState()
                        }
                        "nextState" -> {
                            onNextState()
                        }
                        "skip" -> {
                            onSkip()
                        }
                        "save" -> {
                            onSave()
                        }
                        "load" -> {
                            onLoad()
                        }
                        "play" -> {
                            onPlay()
                        }
                        "changeAlgoState" -> {
                            onChangeAlgoState()
                        }
                        else -> {
                            logger.info("Can't response :(" +
                                    "\nsender = $sender")
                        }
                    }
                }
                else {
                    logger.info("Can't response :(")
                }
```

```
            }
            is VertexUI -> {
                if (isAlgoStarted.value) {
                    logger.info("Start algorithm")
                    ui?.footer?.textState?.value = "Выбрана стартовая
вершина"

                    startAlgorithm(sender)
                }
                else {
                    when (selectedTool) {
                        SelectedTool.REMOVE_VERTEX -> {
                            removeVertex(vertexUI = sender)
                        }
                        SelectedTool.ADD_EDGE -> {
                            if (!vertexSelected.first) {
                                vertexSelected = Pair(true, sender)
                            } else {
                                if (vertexSelected.second != sender) {
                                    addEdge(Pair(vertexSelected.second!!,
sender))

                                    vertexSelected = Pair(false, null)
                                }
                            }
                            logger.info {
                                "Vertex selected: $vertexSelected"
                            }
                        }
                        else -> {}
                    }
                }
            }
            is EdgeUI -> {
                when (selectedTool) {
                    SelectedTool.REMOVE_EDGE -> {
                        removeEdge(sender)
                    }
                    else -> {}
                }
```

```
            }
        }
    }
```

kotlin/application/logic/graph/algorithm.kt

```
    private fun addVertex(offset: Offset, vertex: Vertex = Vertex(),
needToMove: Boolean = true): Boolean {
        // center offset -> top-left offset
        val topLeftOffset = Offset(x = offset.x - VERTEX_SIZE / 2, y =
offset.y - VERTEX_SIZE / 2)
        val vertexUI = VertexUI(vertex = vertex, tools = this)
        if (needToMove) {
            vertexUI.topLeftOffset = topLeftOffset
        }
        else {
            vertexUI.topLeftOffset = offset
        }
        ui!!.graphUI.addVertex(vertexUI)
        verticesAmount.value += 1
//        logger.info("[Tools] Vertex was added with offset: $offset")
        return true
    }

    private fun removeVertex(vertexUI: VertexUI): Boolean {
        return try {
            val removedEdgesAmount = ui!!.graphUI.removeVertex(vertexUI)
            verticesAmount.value -= 1
            edgesAmount.value -= removedEdgesAmount.toInt()
            true
        } catch (exception: Exception) {
            logger.info(exception.message)
            false
        }
    }

    private fun addEdge(verticesUI: Pair<VertexUI, VertexUI>, weight:
UInt = 1u, isDialogOpenOnCreate: Boolean = true): Boolean {
        return try {
            val edgeUI = EdgeUI(
```

```
                verticesUI = verticesUI,
                tools = this,
                edge = Edge(
                    Pair(
                        first = verticesUI.first.vertex,
                        second = verticesUI.second.vertex
                    ),
                    weight = weight.toInt()
                ),
                isDialogOpenOnCreate = isDialogOpenOnCreate
            )
            ui!!.graphUI.addEdge(edgeUI)
            edgesAmount.value += 1
            true
        } catch (exception: Exception) {
            logger.info(exception.message)
            false
        }
    }

    private fun removeEdge(edgeUI: EdgeUI): Boolean {
        return try {
            ui!!.graphUI.removeEdge(edgeUI)
            edgesAmount.value -= 1
            true
        } catch (exception: Exception) {
            logger.info(exception.message)
            false
        }
    }

    private fun onPreviousState() {
        outdatedLookingVertex = null
        if (stateIndex != null) {
            if (stateIndex!! >= 1u) {
//                wasLookedVertices.keys.forEach {
//                    if (it.colorState.value != VertexColor.WAS_LOOKED)
{
//                        it.colorState.value = VertexColor.WAS_LOOKED
```

```
//                      }
//                  }

                val (newColor, newVertexUI) = this.let {
                    val currentVertex = updatesNow?.currentVertexChange
                    val vertexToCost = updatesNow?.markVertexChange

                    if (currentVertex != null) {
                        val vertexUI =
ui?.graphUI?.vertexToUI?.get(currentVertex.second)
                        Pair(VertexColor.WAS_LOOKED, vertexUI)
                    }
                    else if(vertexToCost != null) {
                        val vertexUI =
ui?.graphUI?.vertexToUI?.get(vertexToCost.first)
                        if (vertexToCost.second.first == Int.MAX_VALUE) {
                            vertexUI?.weightInAlgorithmState?.value = "∞"
                            Pair(VertexColor.DEFAULT, vertexUI)
                        }
                        else {
                            vertexUI?.weightInAlgorithmState?.value =
vertexToCost.second.first.toString()
                            Pair(VertexColor.WAS_LOOKED, vertexUI)
                        }
                    }
                    else {
                        Pair(VertexColor.DEFAULT, null)
                    }
                }
                if (newVertexUI != null) {
                    newVertexUI.colorState.value = newColor
                }

                stateIndex = stateIndex!! - 1u
                val (prevColor, prevVertexUI) = this.let {
                    val currentVertex = updatesNow?.currentVertexChange
                    val vertexToCost = updatesNow?.markVertexChange

                    if (currentVertex != null) {
```

```
                        val vertexUI =
ui?.graphUI?.vertexToUI?.get(currentVertex.second)

                        ui?.footer?.textState?.value = "Среди
незафиксированных вершин с весом, отличных от ∞, выбрана вершина с
мин.весом." +
                                "\nВершина зафиксирована с весом $
{vertexUI?.weightInAlgorithmState?.value}"

                        Pair(VertexColor.FIXED, vertexUI)
                    }
                    else if(vertexToCost != null) {
                        val vertexUI =
ui?.graphUI?.vertexToUI?.get(vertexToCost.first)

                        ui?.footer?.textState?.value = "Рассматриваем
нефиксированную вершину среди соседей зафиксированной" +
                                "\nМеняем вес с $
{vertexUI?.weightInAlgorithmState?.value} на $
{vertexToCost.second.second.toString()}"

                        if (vertexUI != null) {
                            outdatedLookingVertex = vertexUI
                        }
                        Pair(VertexColor.LOOKING, vertexUI)
                    }
                    else {
                        newVertexUI?.weightInAlgorithmState?.value = "∞"
                        Pair(VertexColor.DEFAULT, newVertexUI)
                    }
                }

                if (prevVertexUI != null) {
                    prevVertexUI.colorState.value = prevColor
                }

                if (logger.isDebugEnabled) {
                    logger.debug("$updatesNow")
                }
```

```
            }
        }
    }

    private fun onNextState() {
        if (stateIndex != null) {
            if (stateIndex!! + 1u <= if (maxStateIndex != null)
maxStateIndex!! else 0u) {
                stateIndex = stateIndex!! + 1u
                val currentVertex = updatesNow?.currentVertexChange
                val vertexToCost = updatesNow?.markVertexChange

                if (logger.isDebugEnabled) {
                    logger.debug("$updatesNow")
                }

                if(outdatedLookingVertex != null) {
                    outdatedLookingVertex?.colorState?.value =
VertexColor.WAS_LOOKED
                    outdatedLookingVertex = null
                }

                if (currentVertex != null) {
                    val vertexUI =
ui?.graphUI?.vertexToUI?.get(currentVertex.second)
                    if (stateIndex == 1u) {
                        vertexUI?.weightInAlgorithmState?.value = "0"
                    }
                    vertexUI?.colorState?.value = VertexColor.FIXED
                    ui?.footer?.textState?.value = "Среди
незафиксированных вершин с весом, отличных от ∞, выбрана вершина с
мин.весом." +
                            "\nВершина зафиксирована с весом $
{vertexUI?.weightInAlgorithmState?.value}"

                    if (stateIndex == maxStateIndex) {
                        ui?.footer?.textState?.value = "Среди
незафиксированных вершин с весом, отличных от ∞, выбрана вершина с
мин.весом." +
```

```
                                    "\nВершина зафиксирована с весом $
{vertexUI?.weightInAlgorithmState?.value}" +
                                    "\nЗафиксирована последняя вершина.
Алгоритм закончил работу"
                        }
                    }
                    else if(vertexToCost != null) {
                        val vertexUI =
ui?.graphUI?.vertexToUI?.get(vertexToCost.first)
                        if (vertexUI != null) {
                            outdatedLookingVertex = vertexUI

                            ui?.footer?.textState?.value = "Рассматриваем
нефиксированную вершину среди соседей зафиксированной" +
                                    "\nМеняем вес с $
{vertexUI.weightInAlgorithmState.value} на $
{vertexToCost.second.second.toString()}"

                            vertexUI.weightInAlgorithmState.value =
vertexToCost.second.second.toString()
                            vertexUI.colorState.value = VertexColor.LOOKING
                            outdatedLookingVertex = vertexUI
                        }
                    }
                }
            }
        }

    private fun startAlgorithm(sender: VertexUI) {
        dropVerticesWeight()

        if (!isAlgoStarted.value) {
            isAlgoStarted.value = true
        }
        stateMachine = algorithm.dijkstraAlgorithm(graph = logic.graph,
start = sender.vertex)
        stateIndex = 0u

        ui?.graphUI?.verticesUI?.keys?.forEach {
```

```
            it.weightInAlgorithmState.value = "∞"
            it.colorState.value = VertexColor.DEFAULT
        }
    }

    private fun onSkip() {
        if (stateMachine != null)
            loadToStep(maxStateIndex!!)
    }

    private fun onSave() {
        val start =
stateMachine?.getUpdate(1)?.currentVertexChange?.second
        val gfw = GraphFileWriter("test.txt")

        val coords =
            ui?.graphUI?.verticesUI?.keys?.map { it.vertex to
Pair(it.topLeftOffset.x, it.topLeftOffset.y)}?.toMap()?.toMutableMap()

        if (coords == null) {
            logger.info("Trying to save no coords")
        }
        else {
            gfw.toFile(
                fileInfo = FileInfo(
                    graph = logic.graph,
                    start = start,
                    coords = coords,
                    stateNumber = stateIndex?.toInt()
                )
            )
        }
    }

    private fun onLoad() {
        if (!isPlayingNow.value && isPlayingFinished) {
            logger.info("Trying to read the file \"test.txt\"")
            try {
                val gfr = GraphFileReader("test.txt")
```

```
                val fileInfo = gfr.getFileInformation()

                algoReset()

                while(ui?.graphUI?.verticesUI?.size!! > 0) {
                    val vertexUI =
ui?.graphUI?.verticesUI!!.keys.elementAt(0)
                    removeVertex(vertexUI)
                }

                println("fileInfo: $fileInfo")

                fileInfo.coords.forEach {
                    println("${it.key}")
                    addVertex(vertex = it.key, offset = Offset(x =
it.value.first, y = it.value.second), needToMove = false)
                }

                fileInfo.graph.getEdges().forEach {
                    addEdge(
                        Pair(
                            first =
ui!!.graphUI.vertexToUI[it.vertices.first]!!,
                            second =
ui!!.graphUI.vertexToUI[it.vertices.second]!!
                        ),
                        weight = it.weight.toUInt(),
                        isDialogOpenOnCreate = false
                    )
                }

                val stateIndex = fileInfo.stateNumber?.toUInt()

                if (stateIndex != null) {

startAlgorithm(ui?.graphUI?.vertexToUI?.get(fileInfo.start)!!)
                    loadToStep(stateIndex!!)
                }
```

```
                logger.info("Successful read file \"test.txt\"")

            }
            catch(exc: Exception) {
                ui?.footer?.textState?.value = "Ошибка при загрузке
графа"

                logger.error(exc) {
                    "Broken file: ${exc.message}"
                }
            }
        }
        else {
            ui?.footer?.textState?.value = "Прежде, чем загрузить граф,
остановите проигрываение алгоритма"
        }

    }

    private fun onPlay(playDelaySec: UInt = 5u) {
        logger.info("Playing started!")
        if (isPlayingNow.value) {
            isPlayingNow.value = false
        }
        else {
            isPlayingNow.value = true
            val delay = (playDelaySec * 1000u).toLong()

            if(isPlayingFinished) {
                GlobalScope.launch {
                    isPlayingFinished = false
                    while(stateIndex != null && stateIndex !=
maxStateIndex!!) {
                        if (isPlayingNow.value) {
                            onNextState()
                            delay(delay)
                        }
                        else {
                            isPlayingFinished = true
```

```
                        isPlayingNow.value = false
                        this.cancel()
                    }
                }
                if (stateIndex!! == maxStateIndex!!) {
                    isPlayingNow.value = false
                }
                isPlayingFinished = true
                isPlayingNow.value = false
            }
        }
    }
}

private fun algoReset() {
    dropVerticesWeight()
    stateMachine = null
    stateIndex = null

    vertexSelected = Pair(false, null)
    outdatedLookingVertex = null

    isPlayingNow.value = false
    isPlayingFinished = true
}

private fun dropVerticesWeight() {
    ui?.graphUI?.verticesUI?.keys?.forEach {
        it.weightInAlgorithmState.value = ""
        it.colorState.value = VertexColor.DEFAULT
    }
}

private fun loadToStep(step: UInt) {
    while (stateIndex!! != step && stateIndex!! <= maxStateIndex!!) {
        onNextState()
    }
}
```

```
    private fun onChangeAlgoState() {
        isAlgoStarted.value = !isAlgoStarted.value
        if (!isAlgoStarted.value) {
            ui?.footer?.textState?.value = "Задайте граф и запустите
алгоритм"

            algoReset()
        }
        else {
            ui?.footer?.textState?.value = "Запущен алгоритм. Выберите
стартовую вершину"
        }
    }
}
```

kotlin/application/logic/graph/AlgorithmTest.kt

```
package application.logic.graph

import application.logic.VertexInfo
import application.logic.serialization.AlgorithmUpdate
import logic.*
import org.junit.jupiter.api.Test

import org.junit.jupiter.api.Assertions.*
import org.junit.jupiter.api.BeforeEach


internal class AlgorithmTest {
    fun buildGraph(verticesStringFormat: MutableList<String>,
edgesStringFormat: MutableMap<String, MutableMap<String, Int>>): Graph {
        val vertices = verticesStringFormat.map { it to
Vertex(it) }.toMap()

        val outgoingEdges = mutableMapOf<Vertex, MutableList<Pair<Vertex,
Edge>>>()
        val inputgoingEdges = mutableMapOf<Vertex,
MutableList<Pair<Vertex, Edge>>>()

        vertices.values.forEach {
```

```
            outgoingEdges[it] = mutableListOf()
            inputgoingEdges[it] = mutableListOf()
        }

        edgesStringFormat.forEach {
            val it1 = it
            it.value.forEach {
                val pair = Pair(vertices[it1.key]!!, vertices[it.key]!!)
to
                        Edge(
                            vertices = Pair(vertices[it1.key]!!,
vertices[it.key]!!),
                            weight = it.value
                        )

outgoingEdges[pair.first.first]!!.add(Pair(pair.first.second,
pair.second))

inputgoingEdges[pair.first.second]!!.add(Pair(pair.first.first,
pair.second))
            }
        }

        val verticesInfo = mutableMapOf<Vertex, VertexInfo>()
        verticesStringFormat.forEach {
            verticesInfo[vertices[it]!!] = VertexInfo()
        }

        outgoingEdges.forEach {
            val vertex = it.key
            it.value.forEach {
                verticesInfo[vertex]!!.outGoingEdges[it.first] =
it.second
            }
        }

        inputgoingEdges.forEach {
            val vertex = it.key
```

```
            it.value.forEach {
                verticesInfo[vertex]!!.inputGoingEdges[it.first] =
it.second
            }
        }

        val graph = Graph(verticesInfo)

        return graph

    }

    @Test
    fun dijkstraAlgorithm_trivial() {
        val verticesStringFormat = mutableListOf(
            "a", "b", "c","d"
        )

        val edgesStringFormat = mutableMapOf(
            "a" to mutableMapOf(
                "b" to 1,
                "c" to 4,
                "d" to 10
            ),
            "b" to mutableMapOf(
                "c" to 2,
            ),
            "c" to mutableMapOf(
                "d" to 3
            )
        )
        val graph = buildGraph(verticesStringFormat, edgesStringFormat)
        val waitingValues = mutableMapOf<String, String>(
            "a" to "0.0",
            "b" to "1.0",
            "c" to "3.0",
            "d" to "6.0"
        )
        val alg = Algorithm()
```

```
        val stateMachine =
alg.dijkstraAlgorithm(graph,graph.getVertices().first())
        val finalst = stateMachine.getUpdate(0)
        val update = AlgorithmUpdate(null,null)

        assertEquals(finalst,update)
    }
    @Test
    fun dijkstraAlgorithm1_non_trivial() {
        val verticesStringFormat = mutableListOf(
            "A", "B1", "B2","C1","C2","E","I","J"
        )

        val edgesStringFormat = mutableMapOf(
            "A" to mutableMapOf(
                "B1" to 7,
                "E" to 5,
            ),
            "B1" to mutableMapOf(
                "B2" to 8,
                "C1" to 3,
                "C2" to 10
            ),
            "B2" to mutableMapOf(
                "C1" to 3
            ),
            "C1" to mutableMapOf(
                "J" to 20
            ),
            "C2" to mutableMapOf(
                "E" to 1,
                "J" to 6
            ),
            "E" to mutableMapOf(
                "J" to 3
            ),
            "J" to mutableMapOf(
                "I" to 4,
                "B2" to 1
```

```
            ),
            "I" to mutableMapOf(
                "E" to 5
            ),

        )
        val graph = buildGraph(verticesStringFormat, edgesStringFormat)
        val waitingValues = mutableMapOf<String, String>(
            "A" to "0.0",
            "B1" to "7.0",
            "B2" to "9.0",
            "C1" to "10.0",
            "C2" to "17.0",
            "E" to "5.0",
            "J" to "8.0",
            "I" to "12.0",
        )
        val alg = Algorithm()
        val stateMachine =
alg.dijkstraAlgorithm(graph,graph.getVertices().first())
        val finalst = stateMachine.getUpdate(3)

        val v = graph.getVertices().filter { vertex: Vertex ->
vertex.vertexName == "E" }
        val update =
AlgorithmUpdate(null,Pair(v[0],Pair(Float.POSITIVE_INFINITY.toInt(),5)))

        assertEquals(finalst,update)

    }
    @Test
    fun dijkstraAlgorithm_update_doesntexist() {
        val verticesStringFormat = mutableListOf(
            "a", "b", "c","d","e"
        )

        val edgesStringFormat = mutableMapOf(
            "a" to mutableMapOf(
                "b" to 1,
```

```
                "c" to 3,
            ),
            "b" to mutableMapOf(
                "c" to 2,
            ),
            "c" to mutableMapOf(
                "d" to 3
            )
        )
        val graph = buildGraph(verticesStringFormat, edgesStringFormat)
        val waitingValues = mutableMapOf<String, String>(
            "a" to "0.0",
            "b" to "1.0",
            "c" to "3.0",
            "d" to "6.0",
            "e" to "Infinity"
        )
        val alg = Algorithm()
        val stateMachine =
alg.dijkstraAlgorithm(graph,graph.getVertices().first())
        val finalst = stateMachine.getUpdate(99)
        assertNull(finalst.currentVertexChange)
        assertNull(finalst.markVertexChange)

    }
    @Test
    fun dijkstraAlgorithm_no_path() {
        val verticesStringFormat = mutableListOf(
            "a", "b", "c","d","e"
        )

        val edgesStringFormat = mutableMapOf(
            "a" to mutableMapOf(
                "b" to 1,
                "c" to 3,
            ),
            "b" to mutableMapOf(
                "c" to 2,
            ),
```

```
            "c" to mutableMapOf(
                "d" to 3
            )
        )
        val graph = buildGraph(verticesStringFormat, edgesStringFormat)
        val waitingValues = mutableMapOf<String, String>(
            "a" to "0.0",
            "b" to "1.0",
            "c" to "3.0",
            "d" to "6.0",
            "e" to "Infinity"
        )
        val alg = Algorithm()
        val stateMachine =
alg.dijkstraAlgorithm(graph,graph.getVertices().first())
        val finalst = stateMachine.getUpdate(stateMachine.size-1)
        val vC = graph.getVertices().filter { vertex: Vertex ->
vertex.vertexName == "c" }[0]
        val vD = graph.getVertices().filter { vertex: Vertex ->
vertex.vertexName == "d" }[0]
        assertEquals(AlgorithmUpdate(Pair(vC,vD),null),finalst)


    }
    }
```

kotlin/application/logic/graph/edge.kt

```
package logic


class Edge(val vertices: Pair<Vertex, Vertex>, var weight: Int = 0) {
    override fun toString(): String {
        return "${vertices.first} ${vertices.second} $weight"
    }
}
```

kotlin/application/logic/graph/graph.kt

```
package logic

import application.logic.VertexInfo
```

```
class Graph(
    val vertices: MutableMap<Vertex, VertexInfo> = mutableMapOf(),
) {
    fun addVertex(vertex: Vertex) {
        vertices[vertex] = VertexInfo()  //init
    }
    fun addEdge(edge: Edge){
        if(vertices.containsKey(edge.vertices.first) &&
vertices.containsKey(edge.vertices.second) && edge.weight >= 0){

vertices[edge.vertices.first]!!.addOutGoingEdge(edge.vertices.second,
edge)

vertices[edge.vertices.second]!!.addInputEdge(edge.vertices.first, edge)
        }
    }

    private fun removeEdgesTo(vertex:Vertex){
        val toDeleteEdgesInfo = vertices[vertex]

        toDeleteEdgesInfo?.outGoingEdges?.values?.forEach {

vertices[it.vertices.second]?.inputGoingEdges?.remove(it.vertices.first)
        }
        toDeleteEdgesInfo?.outGoingEdges?.clear()

        toDeleteEdgesInfo?.inputGoingEdges?.values?.forEach {

vertices[it.vertices.first]?.outGoingEdges?.remove(it.vertices.second)
        }
        toDeleteEdgesInfo?.inputGoingEdges?.clear()
    }

    fun removeVertex(vertex: Vertex) {
        if (vertices.containsKey(vertex)) {
            removeEdgesTo(vertex)
            vertices.remove(vertex)
```

```
        }
    }

    fun removeEdge(edge: Edge) {
        if
(vertices[edge.vertices.first]?.outGoingEdges?.values?.contains(edge) ==
true){

vertices[edge.vertices.first]?.outGoingEdges?.remove(edge.vertices.second
)

vertices[edge.vertices.second]?.inputGoingEdges?.remove(edge.vertices.fir
st)
        }
    }

    fun getDestinations(vertex:Vertex):MutableSet<Edge>?{
        if(vertex in vertices.keys){
            val dests:MutableSet<Edge> = mutableSetOf()
            vertices[vertex]?.outGoingEdges?.values?.forEach {
                dests.add(it)
            }
          return dests
        }
        return null
    }

    fun getVertices():MutableSet<Vertex>{
        return vertices.keys
    }

    fun getEdges():MutableSet<Edge>{
        val edges:MutableSet<Edge> = mutableSetOf()
        vertices.keys.forEach {
            edges.addAll(vertices[it]!!.outGoingEdges?.values)
        }
        return edges
    }
```

```
}
```

kotlin/application/logic/graph/GraphTest.kt

```
package application.logic.graph

import logic.*
import org.junit.jupiter.api.Assertions
import org.junit.jupiter.api.Assertions.*
import org.junit.jupiter.api.BeforeEach
import org.junit.jupiter.api.Test

internal class GraphTest {
    val graph = Graph()

    @BeforeEach
    fun before() {
        val graph = Graph()
    }

    @Test
    fun addVertex_non_trivial() {

        graph.addVertex(Vertex(""))
        graph.addVertex(Vertex("cde"))
        graph.addVertex(Vertex("dd"))
        val added_vertices = graph.vertices.keys.toString()
        assertEquals("[, cde, dd]", added_vertices)
    }

    @Test
    fun addVertex() {
        graph.addVertex(Vertex("a"))
        graph.addVertex(Vertex("b"))
        graph.addVertex(Vertex("d"))
        val added_vertices = graph.vertices
        assertEquals(3, added_vertices.size)
    }
```

```
    @Test
    fun addVertex_not_correct() {
        val graph = Graph()
        graph.addVertex(Vertex("b"))
        val added_vertices = graph.vertices.keys.toString()
        assertNotEquals("[a]", added_vertices)
    }
    @Test
    fun addEdge_non_trivial() {
        val graph = Graph()
        val v1 = Vertex("a")
        val v2 = Vertex("b")
        graph.addVertex(v1)
        graph.addVertex(v2)
        val v3 = Vertex("c")
        val edge = Edge(Pair(v3, v2), 10)
        val edges = graph.getEdges()
        val exception =
Assertions.assertThrows(IndexOutOfBoundsException::class.java) {
            edges.elementAt(0).toString()
        }
    }

    @Test
    fun addEdge() {
        val graph = Graph()
        val v1 = Vertex("a")
        val v2 = Vertex("b")
        graph.addVertex(v1)
        graph.addVertex(v2)
        graph.addEdge(Edge(Pair(v1, v2), 10))
        val edge1 = graph.getEdges()
        assertEquals("a b 10", edge1.elementAt(0).toString())
    }

    @Test
    fun addEdge_not_correct() {
        val graph = Graph()
        val v1 = Vertex("a")
```

```
    val v2 = Vertex("b")
    graph.addVertex(v1)
    graph.addVertex(v2)
    graph.addEdge(Edge(Pair(v1, v2), 5))
    val edge1 = graph.getEdges()
    assertNotEquals("a b 10", edge1.elementAt(0).toString())
}

@Test
fun removeVertex() {
    val graph = Graph()
    val v1 = Vertex("a")
    graph.addVertex(v1)
    graph.removeVertex(v1)
    val got_one = graph.vertices.keys.toString()
    assertEquals("[]", got_one)

}

@Test
fun removeVertex_not_correct() {
    val graph = Graph()
    val v1 = Vertex("a")
    val v2 = Vertex("b")
    graph.addVertex(v1)
    graph.addVertex(v2)
    graph.removeVertex(v1)
    val got_one = graph.vertices.keys.toString()
    assertNotEquals("[]", got_one)

}
@Test
fun removeVertex_non_trivial() {
    val graph = Graph()
    val v1 = Vertex("a")
    val v2 = Vertex("b")
    graph.addVertex(v1)
    graph.removeVertex(v2)
    val got_one = graph.vertices.keys.toString()
```

```
    assertNotEquals("[]", got_one)

}

@Test
fun removeEdge() {
    val graph = Graph()
    val v1 = Vertex("a")
    val v2 = Vertex("b")
    graph.addVertex(v1)
    graph.addVertex(v2)
    //graph.addEdge(Edge(Pair(v1,v2),10))
    val edge2 = Edge(Pair(v1, v2), 10)
    graph.removeEdge(edge2)
    assertEquals("[]", graph.getEdges().toString())

}
@Test
fun removeEdge_not_correct() {
    val graph = Graph()
    val v1 = Vertex("a")
    val v2 = Vertex("b")
    val v3 = Vertex("c")
    val v4 = Vertex("d")
    graph.addVertex(v1)
    graph.addVertex(v2)
    graph.addVertex(v3)
    graph.addVertex(v4)
    //graph.addEdge(Edge(Pair(v1,v2),10))
    val edge1 = Edge(Pair(v1, v2), 10)
    val edge2 = Edge(Pair(v3, v4), 5)
    graph.addEdge(edge1)
    graph.removeEdge(edge2)
    assertNotEquals("[]", graph.getEdges().toString())

}


@Test
```

```
    fun getDestinations() {
        val vertexA = Vertex("a")
        val vertexB = Vertex("b")
        val vertexC = Vertex("c")
        val vertexD = Vertex("d")
        graph.addVertex(vertexA)
        graph.addVertex(vertexB)
        graph.addVertex(vertexC)
        graph.addVertex(vertexD)
        graph.addEdge(Edge(Pair(vertexA, vertexB), 5))
        graph.addEdge(Edge(Pair(vertexA, vertexC), 4))
        graph.addEdge(Edge(Pair(vertexA, vertexD), 6))
        assertEquals("[a b 5, a c 4, a d
6]",graph.getDestinations(vertexA).toString())
    }
    @Test
    fun getDestinations_not_correct() {
        val vertexA = Vertex("a")
        val vertexB = Vertex("b")
        val vertexC = Vertex("c")
        val vertexD = Vertex("d")
        graph.addVertex(vertexA)
        graph.addVertex(vertexB)
        graph.addVertex(vertexC)
        graph.addVertex(vertexD)
        graph.addEdge(Edge(Pair(vertexA, vertexB), 5))
        graph.addEdge(Edge(Pair(vertexA, vertexC), 4))
        graph.addEdge(Edge(Pair(vertexA, vertexD), 6))
        assertNotEquals("[a b 5, a c
4]",graph.getDestinations(vertexA).toString())
    }

    @Test
    fun getVertices() {
        val graph = Graph()
        val v1 = Vertex("a")
        graph.addVertex(v1)
        assertEquals("[a]", graph.getVertices().toString())
```

```
    }

    @Test
    fun getVertices_not_correct() {
        val graph = Graph()
        val v1 = Vertex("b")
        graph.addVertex(v1)
        assertNotEquals("[a]", graph.getVertices().toString())

    }

    @Test
    fun getEdges() {
        val vertexA = Vertex("a")
        val vertexB = Vertex("b")
        val vertexC = Vertex("c")
        val vertexD = Vertex("d")
        graph.addVertex(vertexA)
        graph.addVertex(vertexB)
        graph.addVertex(vertexC)
        graph.addVertex(vertexD)
        graph.addEdge(Edge(Pair(vertexA, vertexB), 5))
        graph.addEdge(Edge(Pair(vertexA, vertexC), 4))
        graph.addEdge(Edge(Pair(vertexA, vertexD), 6))
        assertEquals(3, graph.getEdges().size)
        val edges = graph.getEdges()
        val expected = mutableListOf<String>(
            "a b 5",
            "a c 4",
            "a d 6"
        )
        for (i in 0..edges.size-1) {
            assertEquals(expected.elementAt(i),
edges.elementAt(i).toString())
        }
    }
    @Test
    fun getEdges_non_correct() {
        val vertexA = Vertex("a")
```

```
        val vertexB = Vertex("b")
        val vertexC = Vertex("c")
        val vertexD = Vertex("d")
        graph.addVertex(vertexA)
        graph.addVertex(vertexB)
        graph.addVertex(vertexC)
        graph.addVertex(vertexD)
        graph.addEdge(Edge(Pair(vertexA, vertexB), 5))
        graph.addEdge(Edge(Pair(vertexA, vertexD), 6))
        assertNotEquals(3, graph.getEdges().size)
        val edges = graph.getEdges()
        val expected = mutableListOf<String>(
            "a b 6",
            "a c 4",
        )
        for (i in 0..edges.size-1) {
            assertNotEquals(expected.elementAt(i),
edges.elementAt(i).toString())
        }
    }
}
```

kotlin/application/logic/graph/priority_vertex.kt

```
package application.logic.graph

import logic.Vertex

data class PriorityVertex(val vertex: Vertex, val priority: Int) :
Comparable<PriorityVertex> {
    override fun compareTo(other: PriorityVertex): Int {
        if (priority > other.priority) return 1
        else if (priority < other.priority) return -1
        return 0
    }
}
package logic
```

kotlin/application/logic/graph/vertex.kt

```
class Vertex() {
    var vertexName = this.hashCode().toString()
```

```
    constructor(name:String) : this() {
        vertexName = name
    }

    override fun toString(): String {
        return vertexName
    }
}
```

kotlin/application/logic/graph/vertex_info.kt

```
package application.logic

import logic.Edge
import logic.Vertex

class VertexInfo(val outGoingEdges:MutableMap<Vertex, Edge> =
mutableMapOf(),
                 val inputGoingEdges:MutableMap<Vertex, Edge> =
mutableMapOf()) {

    fun addOutGoingEdge(vertex:Vertex, edge:Edge){
        outGoingEdges[vertex] = edge
    }
    fun addInputEdge(vertex:Vertex, edge:Edge){
        inputGoingEdges[vertex] = edge
    }

}
```

kotlin/application/logic/serialization/algorithm_update.kt

```
package application.logic.serialization

import logic.Vertex

data class AlgorithmUpdate(
    val currentVertexChange:Pair<Vertex?, Vertex?>?,
    val markVertexChange:Pair<Vertex, Pair<Int, Int>>?
) {}
```

kotlin/application/logic/serialization/file_info.kt

```
package application.logic.serialization
```

```
import logic.Graph
import logic.Vertex

data class FileInfo(val graph:Graph, val start:Vertex?, val
coords:MutableMap<Vertex, Pair<Float, Float>>, val stateNumber:Int?) {
}
```

kotlin/application/logic/serialization/graph_file_reader.kt

```
package logic

import application.logic.serialization.FileInfo
import application.logic.serialization.Parser
import  java.io.File
import java.io.IOException

class GraphFileReader(fileName: String) {
    var fileName = String()
    private val parser = Parser()

    init {
        this.fileName = fileName

    }

    private fun getFileStrings(): MutableList<String> {
        val lineList = mutableListOf<String>()
        val file = File(fileName)
        if(file.canRead())file.useLines {
            for (i in it) {
                if (i.isNotBlank()) lineList.add(i.trim())
            }
        }
        else throw IOException("File not opened")
        return lineList
    }

    @Throws(IOException::class)
    fun getFileInformation(): FileInfo {
```

```
        parser.parse(getFileStrings())
        val graph = Graph()
        val vertexNameConformity: MutableMap<String, Vertex> =
mutableMapOf()
        val coords: MutableMap<Vertex, Pair<Float, Float>> =
mutableMapOf()
        val coordinatesInformation = parser.getCoordsInformation()
        coordinatesInformation.forEach { nameVertex, coordInf ->
            val currentVertex = Vertex(nameVertex)
            graph.addVertex(currentVertex)
            vertexNameConformity[nameVertex] = currentVertex
            coords[currentVertex] = Pair(coordInf.first, coordInf.second)
        }

        val startName = parser.getStartVertexName()
        val start = if(startName == null) null else
vertexNameConformity[startName]

        val edges: HashSet<Pair<String, String>> = hashSetOf()
        for ((source, destination, weight) in
parser.getGraphInformation()) {
            if (edges.contains(
                    Pair(
                        source,
                        destination
                    )
                )
            ) continue //если такое ребро уже есть, то остальные не
учитываем
            //Проверка есть ли новая вершина в наборе соответствий вершин
            if (source !in vertexNameConformity) {
                vertexNameConformity[source] = Vertex(source)
                graph.addVertex(vertexNameConformity[source]!!)
            }
            if (destination !in vertexNameConformity) {
                vertexNameConformity[destination] = Vertex(destination)
                graph.addVertex(vertexNameConformity[destination]!!)
            }
```

```
            graph.addEdge(Edge(Pair(vertexNameConformity[source]!!,
vertexNameConformity[destination]!!), weight))
            edges.add(Pair(source, destination))
            // Проверка добавления координат

            if (!coords.containsKey(vertexNameConformity[source])) {
                throw IOException("Vertex with \"$source\" name don't
have coordinates")
            }
            if (!coords.containsKey(vertexNameConformity[destination])) {
                throw IOException("Vertex with \"$destination\" name
don't have coordinates")
            }

        }

        return FileInfo(graph, start, coords, parser.getStateNumber())
    }
}
```

kotlin/application/logic/serialization/graph_file_writer.kt

```
import application.logic.serialization.FileInfo
import application.logic.serialization.KeyWords
import application.logic.serialization.TextKeyWords
import logic.Edge
import logic.Vertex
import java.io.File
import java.io.IOException
import java.io.PrintWriter

class GraphFileWriter(private val fileName: String){
    private val textKeyWords:TextKeyWords = TextKeyWords()



    private fun edgesToFile(edges: MutableSet<Edge>,
filePrinter:PrintWriter){
        filePrinter.println(textKeyWords.getKeyWordString(KeyWords.GRAPH)
+ textKeyWords.blockStart)
        edges.forEach {
```

```
            filePrinter.println("\t${it.vertices.first.vertexName} $
{it.vertices.second.vertexName} ${it.weight}")
        }
        filePrinter.println(textKeyWords.blockEnd)
    }

    private fun startToFile(vertex:Vertex, filePrinter:PrintWriter){
        filePrinter.println(textKeyWords.getKeyWordString(KeyWords.START)
+"=" + vertex.vertexName)
    }

    private fun coordsToFile(coords:MutableMap<Vertex, Pair<Float,
Float>>?, filePrinter:PrintWriter){

filePrinter.println(textKeyWords.getKeyWordString(KeyWords.COORDS) +
textKeyWords.blockStart)
        coords!!.forEach { vertex, offsetPair ->
            filePrinter.println("\t${vertex.vertexName} $
{offsetPair.first} ${offsetPair.second}")
        }
        filePrinter.println(textKeyWords.blockEnd)
    }

    private fun stateNumberToFile(number: Int, filePrinter: PrintWriter){

filePrinter.println(textKeyWords.getKeyWordString(KeyWords.STATE_NUMBER)
+"="+ number.toString())
    }

    @Throws(IOException::class)
    fun toFile(fileInfo: FileInfo){
        val file = File(fileName)
        val printer = if (file.canWrite()) file.printWriter() else throw
IOException("Can't write in this file")
        if(printer.checkError()) throw IOException("File didn't open")
        edgesToFile(fileInfo.graph.getEdges(), printer)
        if(fileInfo.start != null)
            startToFile(fileInfo.start, printer)
```

```
        if(fileInfo.stateNumber != null)
            stateNumberToFile(fileInfo.stateNumber, printer)
        coordsToFile(fileInfo.coords, printer)
        printer.close()
    }

}
```

kotlin/application/logic/serialization/GraphFileReaderTest.kt

```
package application.logic.serialization

import logic.Edge
import logic.Graph
import logic.GraphFileReader
import logic.Vertex
import org.junit.jupiter.api.Test

import org.junit.jupiter.api.Assertions.*

internal class GraphFileReaderTest {

    @Test
    fun getFileInformation() {
        val graph_info = GraphFileReader("test.txt")
        val grrrr = graph_info.getFileInformation().graph
        val graph1 = Graph()
        val v1 = Vertex("a")
        val v2 = Vertex("b")
        val v3 = Vertex("c")
        graph1.addVertex(v1)
        graph1.addVertex(v2)
        graph1.addVertex(v3)
        graph1.addEdge(Edge(Pair(v1, v2), 1))
        graph1.addEdge(Edge(Pair(v1, v3), 4))
        graph1.addEdge(Edge(Pair(v2, v3), 2))

assertEquals(graph1.vertices.keys.toString(),grrrr.vertices.keys.toString
())
```

```
    }
    @Test
    fun getFileInformation_not_correct() {
        val graph_info = GraphFileReader("test.txt")
        val grrrr = graph_info.getFileInformation().graph
        val graph1 = Graph()
        val v1 = Vertex("a")
        val v2 = Vertex("d")
        val v3 = Vertex("c")
        graph1.addVertex(v1)
        graph1.addVertex(v2)
        graph1.addVertex(v3)
        graph1.addEdge(Edge(Pair(v1, v2), 1))
        graph1.addEdge(Edge(Pair(v1, v3), 4))
        graph1.addEdge(Edge(Pair(v2, v3), 2))

assertNotEquals(graph1.vertices.keys.toString(),grrrr.vertices.keys.toStr
ing())

        //   assertEquals(graph1.)
        //   assertEquals(0,graph_info.getFileInformation().graph)
    }
}
```

kotlin/application/logic/serialization/key_words.kt

```
package application.logic.serialization

enum class KeyWords{
    COORDS,
    GRAPH,
    STATE_NUMBER,
    START
}
```

kotlin/application/logic/serialization/parser.kt

```
package application.logic.serialization
import java.io.IOException
```

```
class Parser{
    private val textKeyWords = TextKeyWords()
    private val keyWords:MutableMap<String, KeyWords> =
textKeyWords.keyWords
    private val keyWordsStartIndexes: MutableMap<KeyWords, Int> =
mutableMapOf()
    private val keyWordsBlocksEnds: MutableMap<KeyWords, Int> =
mutableMapOf()
    private var strings:MutableList<String> = mutableListOf()


    fun parse(strings:MutableList<String>){
        for (i in strings.indices)
            keyWords.keys.forEach {
                if(strings[i].contains(it, ignoreCase = true)){
                    if(!keyWordsStartIndexes.containsKey(keyWords[it]!!))
{
                        keyWordsStartIndexes[keyWords[it]!!] = i
                    }
                    else{
                        throw IOException("Input error \n${i+2}: $
{strings[i]}")
                    }
                }
        }
        if(!keyWordsStartIndexes.containsKey(KeyWords.GRAPH)){
            throw IOException("Graph is not specified")
        }
        if(!keyWordsStartIndexes.containsKey(KeyWords.COORDS)) throw
IOException("Coords not set")

        this.strings = strings
        checkValidGraph()
        checkValidCoords()
        checkValidBlocks()
    }


    fun getStartVertexName():String?{
```

```
        if(keyWordsStartIndexes.containsKey(KeyWords.START)) {
            val startValue =
                strings[keyWordsStartIndexes[KeyWords.START]!!].filter
{ !it.isWhitespace() }.substringAfter("=")
            if (startValue != "=") {
                return startValue
            }
        }
        return null
    }

    fun getStateNumber():Int?{
        if (keyWordsStartIndexes.containsKey(KeyWords.STATE_NUMBER)){
            val stepValue=
strings[keyWordsStartIndexes[KeyWords.STATE_NUMBER]!!].
            filter { !it.isWhitespace() }.substringAfter("=")
            if (stepValue != "=" && stepValue.toIntOrNull() != null &&
stepValue.toInt() >= 0) return stepValue.toInt()
        }
        return null
    }


    private fun checkValidGraph(){
        var currentIndexString:Int = keyWordsStartIndexes[KeyWords.GRAPH]
!!+ 1
        while (true){
            val splitString = strings[currentIndexString].split("
").filter { s: String -> s.isNotBlank() }
            if (splitString.size == 3) {
                var (source, destination, weight) = splitString
                if (weight.toIntOrNull() == null || weight.toInt() < 0)
                    throw IOException("Invalid edge weight\n$
{currentIndexString+2}: ${strings[currentIndexString]}")
                ++currentIndexString
            }
            else break
        }
```

```
        if(strings[currentIndexString] != textKeyWords.blockEnd) throw
IOException("Wrong graph block")
        keyWordsBlocksEnds[KeyWords.GRAPH] = currentIndexString - 1
    }

    private fun checkValidCoords(){
        var currentIndexString:Int =
keyWordsStartIndexes[KeyWords.COORDS] !!+ 1
        val start = getStartVertexName()

        while (true){
            val splitString = strings[currentIndexString].split('(', ')',
',', ' ').
            filter { s: String -> s.isNotBlank()}

            if (splitString.size == 3) {
                var (vertex, offsetX, offsetY) = splitString
                if (offsetX.toFloatOrNull() == null)
                    throw IOException("Invalid Xoffset\n$
{currentIndexString + 1}: ${strings[currentIndexString]}")
                if (offsetY.toFloatOrNull() == null)
                    throw IOException("Invalid Yoffset\n$
{currentIndexString + 1}: ${strings[currentIndexString]}")
                ++currentIndexString
            }
            else break
        }

        if(strings[currentIndexString] != textKeyWords.blockEnd) throw
IOException("Wrong graph block")
        keyWordsBlocksEnds[KeyWords.COORDS] = currentIndexString - 1
    }

    private fun checkValidBlocks(){
        val graphRange =
keyWordsStartIndexes[KeyWords.GRAPH]!!.rangeTo(keyWordsBlocksEnds[KeyWord
s.GRAPH]!!)
        //Start key check
```

```
        if (keyWordsStartIndexes.containsKey(KeyWords.START) &&
keyWordsStartIndexes[KeyWords.START]!! in graphRange)throw
IOException("Input error")

        val coordsRange = keyWordsBlocksEnds[KeyWords.COORDS]?.let
{ keyWordsStartIndexes[KeyWords.COORDS]?.rangeTo(it) }
        if (coordsRange != null){
            if (keyWordsStartIndexes.containsKey(KeyWords.START) &&
keyWordsStartIndexes[KeyWords.START] in coordsRange)throw
IOException("Input error")
        }
        if(keyWordsStartIndexes.containsKey(KeyWords.STATE_NUMBER)){
            if (coordsRange != null){
                if (keyWordsStartIndexes[KeyWords.STATE_NUMBER]!! in
coordsRange)throw IOException("Input error")
            }
            if (keyWordsStartIndexes[KeyWords.STATE_NUMBER]!! in
graphRange)throw IOException("Input error")
        }
    }

    fun getGraphInformation():MutableList<Triple<String, String, Int>>{
        val graphInformation:MutableList<Triple<String, String, Int>> =
mutableListOf()
        for (i in (keyWordsStartIndexes[KeyWords.GRAPH]!! +
1).rangeTo(keyWordsBlocksEnds[KeyWords.GRAPH]!!)){
            val (source, destination, weight) = strings[i].split('
').filter { s: String ->  s.isNotBlank()}
            graphInformation.add(Triple(source, destination,
weight.toInt()))
        }
        return graphInformation
    }

    fun getCoordsInformation():MutableMap<String, Pair<Float, Float>>{
        val coordsInformation:MutableMap<String, Pair<Float, Float>> =
mutableMapOf()
        for (i in (keyWordsStartIndexes[KeyWords.COORDS]!!
+1).rangeTo(keyWordsBlocksEnds[KeyWords.COORDS]!!)){
```

```
            val (vertex, xOffset, yOffset) = strings[i].split(' ', '(',
')', ',').filter { s: String ->  s.isNotBlank()}
            coordsInformation[vertex] = Pair(xOffset.toFloat(),
yOffset.toFloat())
        }
        return coordsInformation
    }
}
```

kotlin/application/logic/serialization/ParserTest.kt

```
package application.logic.serialization

import org.junit.jupiter.api.Assertions.*
import org.junit.jupiter.api.Test

internal class ParserTest {

    @Test
    fun parse() {
        val string_ingoing = mutableListOf<String>(
            "graph{",
            "a b 1",
            "b c 2",
            "a c 3",
            "b e 2",
            "}",
            "start = a",
            "state = 0",
            "coords{",
            "a(0,0)",
            "b(0,1)",
            "c(1,1)",
            "e(1,2)",
            "}",
        )
        val v = Parser()
        assertDoesNotThrow { v.parse(string_ingoing) }
    }
```

```
@Test
fun parse1() {
    val string_ingoing = mutableListOf<String>(
        "graph{",
        "a b 7",
        "b c 10",
        "c e 1",
        "a e 5",
        "i e 5",
        "a e 15",
        "j i 4",
        "c j 6",
        "}",
        "start = a",
        "state = 3",
        "coords{",
        "a(0,0)",
        "b(0,1)",
        "c(1,1)",
        "e(1,2)",
        "i(2,2)",
        "j(3,1)",
        "}",
    )
    val v = Parser()
    assertDoesNotThrow { v.parse(string_ingoing) }
}
@Test
fun parse2() {
    val string_ingoing = mutableListOf<String>(
        "state = 0",
        "graph{",
        "a b 1",
        "b c 2",
        "c e 1",
        "a e 5",
        "i e 5",
        "a e 15",
        "}",
```

```
            "start = a",
            "coords{",
            "a(0,0)",
            "b(0,1)",
            "c(1,1)",
            "e(1,2)",
            "i(2,2)",
            "}",
        )
        val v = Parser()
        assertDoesNotThrow { v.parse(string_ingoing) }
    }
    @Test
    fun parse3() {
        val string_ingoing = mutableListOf<String>(
            "graph{",
            "a b 1",
            "b c 2",
            "c e 1",
            "a e 5",
            "i e 5",
            "a e 15",
            "}",
            "state = 0",
            "start = a",
            "coords{",
            "a(1,1)",
            "b(2,2)",
            "c(2,3)",
            "}",
        )


        val v = Parser()
        assertDoesNotThrow { v.parse(string_ingoing) }
    }
}
```

kotlin/application/logic/serialization/state_machine.kt

```
package logic

import application.logic.serialization.AlgorithmUpdate


class StateMachine(graph: Graph, start: Vertex) {
    private val states: MutableMap<Vertex, MutableList<Float>> =
mutableMapOf()
    private val currentStateVertexes: MutableList<Vertex?> =
mutableListOf()
    private val updates:MutableList<Pair<Vertex, Int>?>  =
mutableListOf()
    var size = 0
        private set

    init {
        graph.vertices.keys.forEach {
            states[it] = mutableListOf()
            states[it]!!.add(Float.POSITIVE_INFINITY)
        }
        currentStateVertexes.add(null)
        updates.add(null)
    }
    fun getUpdate(index:Int):AlgorithmUpdate {
        return when(index) {
            0 -> AlgorithmUpdate(null, null)
            1 -> AlgorithmUpdate(Pair(null, currentStateVertexes[1]),
null)
            in 2 until size -> {
                val prevState = getState(index)
                val currentVertexChange =
                    if(currentStateVertexes[index] ==
currentStateVertexes[index+1])
                        null
                    else
                        Pair(currentStateVertexes[index],
currentStateVertexes[index+1])
                val markVertexChange =
```

```
                    if(prevState != null && updates[index+1]?.first !=
null && updates[index+1]?.second != null)
                        Pair(updates[index+1]!!.first,
Pair(prevState[updates[index+1]?.first]!!.toInt(),
updates[index+1]!!.second))
                    else null
                AlgorithmUpdate(currentVertexChange, markVertexChange)
            }
            else -> {AlgorithmUpdate(null, null)}
        }
    }
    fun addNextState(currentUpdate: Pair<Vertex, Int>?,
currentStateVertex: Vertex) {
        currentStateVertexes.add(currentStateVertex)
        updates.add(currentUpdate)
        states.keys.forEach {
            states[it]!!.add(states[it]!![size])
        }
        if(currentUpdate != null) {
            states[currentUpdate.first]!!.removeAt(size)

states[currentUpdate.first]!!.add(currentUpdate.second.toFloat())
        }
        ++size
    }

    private fun getState(stateNumber:Int):MutableMap<Vertex, Float>?{
        if(stateNumber in 0 until size){
            val state: MutableMap<Vertex, Float> = mutableMapOf()
            states.keys.forEach {
                state[it] = states[it]!![stateNumber]
            }
            return state
        }
        return null
    }
}
```

kotlin/application/logic/serialization/text_key_words.kt

```
package application.logic.serialization

class TextKeyWords{
    val keyWords:MutableMap<String, KeyWords> = mutableMapOf(
        "coords" to KeyWords.COORDS,
        "graph" to KeyWords.GRAPH,
        "state" to KeyWords.STATE_NUMBER,
        "start" to KeyWords.START)
    val blockStart = "{"

    val blockEnd = "}"

    fun getKeyWordString(key:KeyWords):String{
        for ((stringKey, enumKey) in keyWords){
            if (key == enumKey) return stringKey
        }
        return "NOT FOUND KEY-WORD STRING"
    }

}
```

kotlin/application/logic/logic.kt

```
package application.logic

import logic.Graph

class Logic(val graph: Graph = Graph())
```

UI

kotlin/application/ui/dialog/edge_dialog.kt

```
package application.ui.dialog

import androidx.compose.foundation.layout.padding
import androidx.compose.material.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.MutableState
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.ExperimentalComposeUiApi
```

```
import androidx.compose.ui.Modifier
import androidx.compose.ui.input.key.Key
import androidx.compose.ui.input.key.key
import androidx.compose.ui.unit.dp
import androidx.compose.ui.window.Dialog
import logger

class EdgeDialog {
    val textState = mutableStateOf("1")

    @OptIn(ExperimentalComposeUiApi::class)
    @Composable
    fun draw(isDialogOpen: MutableState<Boolean>, switcher:
MutableState<Boolean>) {
        println("TEXTSTATE: ${textState.value}")
        val text = remember { textState }
        val isDialogOpen = remember { isDialogOpen }
        if (isDialogOpen.value) {
            Dialog(
                onCloseRequest = {
                    isDialogOpen.value = false
                },
                onKeyEvent = {
                    when(it.key) {
                        Key.Escape, Key.Enter -> {
                            isDialogOpen.value = false
                            true
                        }
                        else -> {
                            false
                        }
                    }
                },
                title = "Enter edge weight",
                resizable = false
            ) {
                TextField(
                    modifier = Modifier.padding(50.dp, 75.dp),
                    value = text.value,
```

```
                    onValueChange = {
                        try {
                            if (it.length <= 3) {
                                it.toUInt()
                                text.value = it
                                switcher.value = !switcher.value
                            }
                        }
                        catch (exc: Exception) {
                            logger.error("${exc.message}")
                        }
                    }
                )
            }
        }
    }
}
```

kotlin/application/ui/objects/edge_shape.kt

```
package application.ui.objects

import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.geometry.Size
import androidx.compose.ui.graphics.Outline
import androidx.compose.ui.graphics.Path
import androidx.compose.ui.graphics.Shape
import androidx.compose.ui.unit.Density
import androidx.compose.ui.unit.LayoutDirection
import logger
import kotlin.math.*

class EdgeShape(
    private val startQuarter: Int,
    private val start: Offset,
    private val end: Offset,
    private val lineWidth: Float,
    private val arrowWidth: Float = 10f,
    private val arrowHeight: Float = 10f
): Shape {
```

```
    @Throws(IllegalStateException::class)
    override fun createOutline(
        size: Size,
        layoutDirection: LayoutDirection,
        density: Density
    ): Outline {
        val path = Path().apply {
            reset()
            logger.debug(
                    "\nstart = $start" +
                    "\nend = $end" +
                    "\nsize = $size"
            )

            // SE - start and end
            val lengthBetweenSE = sqrt((start.x - end.x).pow(2) +
(start.y - end.y).pow(2))

            val dots = mutableMapOf(
                "rectLeftTop" to Offset(x = start.x, y = start.y -
lineWidth / 2),
                "rectRightTop" to Offset(x = start.x + lengthBetweenSE -
arrowWidth, y = start.y - lineWidth / 2),
                "arrowLeftTop" to Offset(x = start.x + lengthBetweenSE -
arrowWidth, y = start.y - lineWidth / 2 - arrowHeight),
                "arrowRightCenter" to Offset(x = start.x +
lengthBetweenSE, y = start.y),
                "arrowLeftBottom" to Offset(x = start.x + lengthBetweenSE
- arrowWidth, y = start.y + lineWidth / 2 + arrowHeight),
                "rectRightBottom" to Offset(x = start.x + lengthBetweenSE
- arrowWidth, y = start.y + lineWidth / 2),
                "rectLeftBottom" to Offset(x = start.x, y = start.y +
lineWidth / 2)
            )

            val tan = abs(end.y - start.y) / abs(end.x - start.x)

            val degToRad = { angle: Float -> (angle * Math.PI /
180f).toFloat()}
```

```
            val angle = when(startQuarter) {
                1 -> { degToRad(180f) + atan(-tan) }
                2 -> { atan(tan) }
                3 -> { atan(-tan) }
                4 -> { degToRad(180f) + atan(tan) }
                else -> { throw Exception("Wrong startQuarter") }
            }

            logger.debug(
                "\ntan = $tan" +
                "\nangle = $angle" +
                "\nstartQuarter = $startQuarter"
            )

            // rotate near dot start
            val rotate = { dot: Offset, angle: Float ->
                Offset(
                    x = (dot.x - start.x) * cos(angle) - (dot.y -
start.y) * sin(angle) + start.x,
                    y = (dot.x - start.x) * sin(angle) + (dot.y -
start.y) * cos(angle) + start.y
                )
            }

            val dotsAfterRotation = mutableMapOf(
                "rectLeftTop" to rotate(dots["rectLeftTop"]!!, angle),
                "rectRightTop" to rotate(dots["rectRightTop"]!!, angle),
                "arrowLeftTop" to rotate(dots["arrowLeftTop"]!!, angle),
                "arrowRightCenter" to rotate(dots["arrowRightCenter"]!!,
angle),
                "arrowLeftBottom" to rotate(dots["arrowLeftBottom"]!!,
angle),
                "rectRightBottom" to rotate(dots["rectRightBottom"]!!,
angle),
                "rectLeftBottom" to rotate(dots["rectLeftBottom"]!!,
angle),
            )
```

```
            logger.debug(
                    "\nWithoutRotation:" +
                    "\nrectLeftTop: ${dots["rectLeftTop"]}" +
                    "\nrectRightTop: ${dots["rectRightTop"]}" +
                    "\narrowLeftTop: ${dots["arrowLeftTop"]}" +
                    "\narrowRightCenter: ${dots["arrowRightCenter"]}" +
                    "\narrowLeftBottom: ${dots["arrowLeftBottom"]}" +
                    "\nrectRightBottom: ${dots["rectRightBottom"]}" +
                    "\nrectLeftBottom: ${dots["rectLeftBottom"]}" +
                    "\nWithRotation:" +
                    "\nrectLeftTop: ${dotsAfterRotation["rectLeftTop"]}"
+
                    "\nrectRightTop: $
{dotsAfterRotation["rectRightTop"]}" +
                    "\narrowLeftTop: $
{dotsAfterRotation["arrowLeftTop"]}" +
                    "\narrowRightCenter: $
{dotsAfterRotation["arrowRightCenter"]}" +
                    "\narrowLeftBottom: $
{dotsAfterRotation["arrowLeftBottom"]}" +
                    "\nrectRightBottom: $
{dotsAfterRotation["rectRightBottom"]}" +
                    "\nrectLeftBottom: $
{dotsAfterRotation["rectLeftBottom"]}"
            )

            moveTo(dotsAfterRotation["rectLeftTop"]!!.x,
dotsAfterRotation["rectLeftTop"]!!.y)
            lineTo(dotsAfterRotation["rectRightTop"]!!.x,
dotsAfterRotation["rectRightTop"]!!.y)
            lineTo(dotsAfterRotation["arrowLeftTop"]!!.x,
dotsAfterRotation["arrowLeftTop"]!!.y)
            lineTo(dotsAfterRotation["arrowRightCenter"]!!.x,
dotsAfterRotation["arrowRightCenter"]!!.y)
            lineTo(dotsAfterRotation["arrowLeftBottom"]!!.x,
dotsAfterRotation["arrowLeftBottom"]!!.y)
            lineTo(dotsAfterRotation["rectRightBottom"]!!.x,
dotsAfterRotation["rectRightBottom"]!!.y)
```

```
            lineTo(dotsAfterRotation["rectLeftBottom"]!!.x,
dotsAfterRotation["rectLeftBottom"]!!.y)
            lineTo(dotsAfterRotation["rectLeftTop"]!!.x,
dotsAfterRotation["rectLeftTop"]!!.y)

            close()
        }
        return Outline.Generic(path)
    }
}
```

kotlin/application/ui/objects/edge_ui.kt

```
package application.ui.objects

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.offset
import androidx.compose.foundation.layout.requiredSize
import androidx.compose.foundation.layout.size
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import application.controller.Tools
import application.ui.dialog.EdgeDialog
import logger
import logic.Edge
import kotlin.math.abs
import kotlin.math.min
import kotlin.math.pow
import kotlin.math.sqrt

const val LINE_WIDTH = 10f
```

```
class EdgeUI(
    val verticesUI: Pair<VertexUI, VertexUI>,
    private val tools: Tools,
    val edge: Edge = Edge(Pair(verticesUI.first.vertex,
verticesUI.second.vertex)),
    val isDialogOpenOnCreate: Boolean = true
) {
    val edgeDialog = EdgeDialog()
    @Composable
    fun draw() {
        val size = Offset(
            x = abs(
                verticesUI.first.topLeftOffset.x
                        - verticesUI.second.topLeftOffset.x
            ) + VERTEX_SIZE,
            y = abs(
                verticesUI.first.topLeftOffset.y
                        - verticesUI.second.topLeftOffset.y
            ) + VERTEX_SIZE
        )
        val startTopLeft = Pair(
            first = verticesUI.first.topLeftOffset.x <
verticesUI.second.topLeftOffset.x,
            second = verticesUI.first.topLeftOffset.y <
verticesUI.second.topLeftOffset.y
        )
        val startQuarter = if (startTopLeft.first) {
            if (startTopLeft.second) 2
            else 3
        } else {
            if (startTopLeft.second) 1
            else 4
        }
        // for start position of the vertex with shape circle
        // additional else for case where vertex1.x == vertex2.x
        val dislocation = if (size.x - VERTEX_SIZE != 0f) {
            // gradient = a, there y = ax + b
```

```
            val gradient = (size.y - VERTEX_SIZE) / (size.x -
VERTEX_SIZE)
            val dislocation = let {
                val radius = VERTEX_SIZE / 2
                val x = radius / sqrt((gradient.pow(2) + 1))
                Offset(
                    x = x,
                    y = gradient * x
                )
            }

            logger.info("gradient: $gradient")

            dislocation
        } else Offset(0f, VERTEX_SIZE / 2f)

        logger.info(
            """
                size: ${size.x} ${size.y}
                dislocation: ${dislocation.x} ${dislocation.y}
            """.trimIndent()
        )
        val start =
            when (startQuarter) {
                1 -> Offset(size.x - dislocation.x - VERTEX_SIZE / 2, 0f
+ dislocation.y + VERTEX_SIZE / 2)
                2 -> Offset(0f + dislocation.x + VERTEX_SIZE / 2, 0f +
dislocation.y + VERTEX_SIZE / 2)
                3 -> Offset(0f + dislocation.x + VERTEX_SIZE / 2, size.y
- dislocation.y - VERTEX_SIZE / 2)
                4 -> Offset(size.x - dislocation.x - VERTEX_SIZE / 2,
size.y - dislocation.y - VERTEX_SIZE / 2)
                else -> throw Error("Incorrect startQuarter value")
            }
        val end =
            when (startQuarter) {
                1 -> Offset(0f + dislocation.x + VERTEX_SIZE / 2, size.y
- dislocation.y - VERTEX_SIZE / 2)
```

```
                2 -> Offset(size.x - dislocation.x - VERTEX_SIZE / 2,
size.y - dislocation.y - VERTEX_SIZE / 2)
                3 -> Offset(size.x - dislocation.x - VERTEX_SIZE / 2, 0f
+ dislocation.y + VERTEX_SIZE / 2)
                4 -> Offset(0f + dislocation.x + VERTEX_SIZE / 2, 0f +
dislocation.y + VERTEX_SIZE / 2)
                else -> throw Error("Incorrect startQuarter value")
            }

        val offset = Offset(
            x = min(
                verticesUI.first.topLeftOffset.x,
                verticesUI.second.topLeftOffset.x,
            ),
            y = min(
                verticesUI.first.topLeftOffset.y,
                verticesUI.second.topLeftOffset.y,
            )
        )

        Surface(
            modifier = Modifier
                .offset(
                    x = offset.x.dp,
                    y = offset.y.dp
                )
                .requiredSize(
                    width = size.x.dp,
                    height = size.y.dp
                )
                .clip(EdgeShape(startQuarter, start, end, LINE_WIDTH))
                .clickable {
                    tools.notifyMe(this)
                },
//              shape = EdgeShape(startQuarter, start, end, LINE_WIDTH),
            color = Color.Red
        ) {}

        logger.debug("Offset: $offset")
```

```
        // Imagine edge ------->. weightPos - near witch part of edge
weight will be drawn
        // Must be in range: (0; 1]
        val weightPos = 2 / 3f

        val centerEdgeOffset = when(startQuarter) {
            1 -> {
                Offset(
                    x = offset.x + start.x - weightPos * abs(start.x -
end.x),
                    y = offset.y + start.y + weightPos * abs(start.y -
end.y)
                )
            }
            2 -> {
                Offset(
                    x = offset.x + start.x + weightPos * abs(start.x -
end.x),
                    y = offset.y + start.y + weightPos * abs(start.y -
end.y)
                )
            }
            3 -> {
                Offset(
                    x = offset.x + start.x + weightPos * abs(start.x -
end.x),
                    y = offset.y + start.y - weightPos * abs(start.y -
end.y)
                )
            }
            4 -> {
                Offset(
                    x = offset.x + start.x - weightPos * abs(start.x -
end.x),
                    y = offset.y + start.y - weightPos * abs(start.y -
end.y)
                )
            }
```

```
            else -> {throw Error("Incorrect startQuarter value")}
        }

        val switcher = remember { mutableStateOf(false) }
        val isDialogOpen = remember
{ mutableStateOf(isDialogOpenOnCreate) }
        logger.debug("START: $start, END: $end, CENTER:
$centerEdgeOffset")

        logger.debug("edgeDialog.text: ${edgeDialog.textState.value}")
        Text(
            text = edgeDialog.textState.value,
            color = Color.Black,
            fontSize = 18.sp,
            modifier = Modifier
                .offset(x = centerEdgeOffset.x.dp, y =
centerEdgeOffset.y.dp)
                .clickable {
                    if (!tools.isAlgoStarted.value) {
                        isDialogOpen.value = true
                    }
                }
                .size(36.dp, 24.dp)
        )
        if (isDialogOpen.value == true) {
            edgeDialog.draw(isDialogOpen, switcher)
        }
        edge.weight = edgeDialog.textState.value.toInt()
        logger.debug("startTL: ${verticesUI.first.topLeftOffset}, endTL:
${verticesUI.second.topLeftOffset}")
    }
}
```

kotlin/application/ui/objects/vertex_info_ui.kt

```
package application.ui.objects

import androidx.compose.ui.graphics.Color

enum class VertexColor(val color: Color) {
```

```
    DEFAULT(Color.Blue),
    FIXED(Color.Green),
    WAS_LOOKED(Color.Magenta),
    LOOKING(Color.Yellow)
}

class VertexInfoUI {
    val outGoingEdges:MutableMap<VertexUI, EdgeUI> = mutableMapOf()
    val inputGoingEdges:MutableMap<VertexUI, EdgeUI> = mutableMapOf()

    fun addOutGoingEdge(vertex:VertexUI, edge:EdgeUI){
        outGoingEdges[vertex] = edge
    }
    fun addInputEdge(vertex:VertexUI, edge:EdgeUI){
        inputGoingEdges[vertex] = edge
    }

    fun getOutGoingEdges():MutableList<EdgeUI>{
        return outGoingEdges.values.toMutableList()
    }
    fun getInputEdge():MutableList<EdgeUI>{
        return inputGoingEdges.values.toMutableList()
    }
}
```

kotlin/application/ui/objects/vertex_ui.kt

```
package application.ui.objects

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.offset
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
```

```
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import application.controller.Tools
import logic.Vertex

const val VERTEX_SIZE = 100

class VertexUI(val vertex: Vertex = Vertex(), var topLeftOffset: Offset =
Offset(0f, 0f), private val tools: Tools) {
    var weightInAlgorithmState = mutableStateOf("")
    var colorState = mutableStateOf(VertexColor.DEFAULT)
    @Composable
    fun draw() {
        var weightInAlgorithm = remember { weightInAlgorithmState }
        var color = remember { colorState }
        Text(
            text = weightInAlgorithm.value,
            color = Color.Black,
            fontSize = 26.sp,
            modifier = Modifier
                .fillMaxWidth()
                .fillMaxHeight()
                .offset(x = (topLeftOffset.x + VERTEX_SIZE / 4).dp, y =
(topLeftOffset.y - 3 * VERTEX_SIZE / 4).dp)
                .size((VERTEX_SIZE / 2).dp, (VERTEX_SIZE / 2).dp)
        )
        Surface(
            modifier = Modifier
                .offset(x = topLeftOffset.x.dp, y = topLeftOffset.y.dp)
                .size(VERTEX_SIZE.dp, VERTEX_SIZE.dp)
                .clip(shape = CircleShape)
                .clickable {
                    tools.notifyMe(sender = this)
                },
            shape = CircleShape,
```

```
            color = color.value.color,
        ) {
            Text(
                text = let {
                    try {
                        vertex.vertexName.toInt()
                        ""
                    }
                    catch(exc: Exception) {
                        vertex.vertexName
                    }
                },
                color = Color.Black,
                fontSize = 18.sp,
                modifier = Modifier
                    .fillMaxWidth()
                    .fillMaxHeight()
                    .offset(x = (VERTEX_SIZE / 4).dp, y = (VERTEX_SIZE /
4).dp,)
                    .size((VERTEX_SIZE / 2).dp, (VERTEX_SIZE / 2).dp)
            )
        }
    }
}
```

application/ui/window/canvas.kt

```
package application.ui.window

import androidx.compose.foundation.background
import androidx.compose.foundation.gestures.detectTapGestures
import androidx.compose.foundation.gestures.detectTransformGestures
import androidx.compose.foundation.layout.Box
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.ExperimentalComposeUiApi
import androidx.compose.ui.Modifier
import androidx.compose.ui.composed
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.graphics.Color
```

```
import androidx.compose.ui.graphics.graphicsLayer
import androidx.compose.ui.input.pointer.PointerEventType
import androidx.compose.ui.input.pointer.onPointerEvent
import androidx.compose.ui.input.pointer.pointerInput
import application.controller.Tools
import logger

class Canvas(val tools: Tools) {
    private fun getSuper() = this
    @OptIn(ExperimentalComposeUiApi::class)
    @Composable
    fun draw(modifier: Modifier) {
        val verticesAmount = remember { tools.verticesAmount }
        val edgesAmount = remember { tools.edgesAmount }
        val canvasOffset = remember { mutableStateOf(Offset.Zero) }
        var oldScale = 1f
        val scale = remember { mutableStateOf(1f) }
        Box(
            modifier = Modifier
                .onPointerEvent(PointerEventType.Scroll) {
                    val sign = if (it.changes.first().scrollDelta.y > 0)
"+" else "-"
                    val zoom = 0.1f
                    if (sign == "+" && scale.value >= 0.1f) {
                        oldScale = scale.value
                        scale.value -= zoom
                    }
                    else if (sign == "-" && scale.value < 3f) {
                        oldScale = scale.value
                        scale.value += zoom
                    }
                    logger.info("[Tools] scale: ${scale.value}")
                }
                .background(color = Color.White)
                .pointerInput(Unit) {
                    detectTransformGestures { centroid, pan, zoom,
rotation ->
                        canvasOffset.value -= pan
                    }
```

```
                }
                .pointerInput(Unit) {
                    detectTapGestures { offset ->
                        val centerOffset = tools.ui?.centerCanvasSize
                        val realTapOffset = (offset - centerOffset!! +
canvasOffset.value) / scale.value + centerOffset
                        logger.info("detectTapGestures: realTapOffset =
$realTapOffset" +
                                "\ndetectTapGestures: canvasOffset = $
{canvasOffset.value}")
                        tools.notifyMe(
                            Pair(
                                first = getSuper(),
                                second = realTapOffset
                            )
                        )
                    }
                }
                .graphicsLayer {
                    scaleX = scale.value
                    scaleY = scale.value
                    translationX = -canvasOffset.value.x
                    translationY = -canvasOffset.value.y
                }
                .composed { modifier }
        ) {

            if (verticesAmount.value > 0) {
                for (vertexUI in tools.ui?.graphUI?.verticesUI!!.keys) {
                    vertexUI.draw()
                }
            }

            if (edgesAmount.value > 0) {
                for (edgeUI in tools.ui?.graphUI?.getEdges()!!) {
                    edgeUI.draw()
                }
            }
            logger.info("verticesAmount: $verticesAmount" +
```

```
                    "\nedgesAmount: $edgesAmount")
        }
    }
}
```

kotlin/application/ui/window/footer.kt

```
package application.ui.window

import androidx.compose.foundation.background
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.composed
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.sp
import application.controller.Tools

class Footer(val tools: Tools) {
    val textState = mutableStateOf("Задайте граф и запустите алгоритм")
    @Composable
    fun draw(modifier: Modifier) {
        val text = remember { textState }
        Text(
            text = textState.value,
            textAlign = TextAlign.Center,
            color = Color.Black,
            fontSize = 30.sp,
            modifier = Modifier
                .composed { modifier }
                .background(color = Color.Gray)
        )
    }
}
```

kotlin/application/ui/ui.kt

```
package application.ui

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.DpSize
import androidx.compose.ui.unit.dp
import androidx.compose.ui.window.WindowState
import androidx.compose.ui.window.singleWindowApplication
import androidx.compose.ui.zIndex
import application.controller.Tools
import application.ui.objects.GraphUI
import application.ui.window.Canvas
import application.ui.window.Footer
import application.ui.window.Toolbar

class UI(private val tools: Tools) {
    private val windowState = WindowState(size = DpSize(600.dp, 600.dp))
    val centerCanvasSize: Offset
        get() = Offset(
            x = windowState.size.width.value * (1f - toolbarWidth) / 2f,
            y = windowState.size.height.value * canvasHeight / 2f
        )
    val graphUI = GraphUI(tools.logic.graph)
    val toolbar = Toolbar(tools)
    val canvas = Canvas(tools)
    val footer = Footer(tools)

    private val toolbarWidth = 0.15f
    private val canvasHeight = 0.75f
    fun draw() {
        singleWindowApplication(
            state = windowState,
            title = "Graph algo solver",
            onKeyEvent = {
                tools.notifyMe(Pair(this, it))
                true
```

```
    }
) {
    Row {
        toolbar.draw(
            modifier = Modifier
                .fillMaxHeight()
                .weight(toolbarWidth, true)
                .zIndex(2f)
        )
        Spacer(
            modifier = Modifier
                .fillMaxHeight()
                .weight(0.00125f, true)
                .background(color = Color.Black)
                .zIndex(2f)
        )
        Column(
            modifier = Modifier
                .fillMaxHeight()
                .weight(1f - toolbarWidth, true)
                .zIndex(1f)
        ) {
            canvas.draw(
                modifier = Modifier
                    .fillMaxWidth()
                    .weight(canvasHeight, true)
                    .zIndex(1f)
            )
            Spacer(
                modifier = Modifier
                    .fillMaxWidth()
                    .weight(0.0025f, true)
                    .background(color = Color.Black)
                    .zIndex(2f)
            )
            footer.draw(
                modifier = Modifier
                    .fillMaxWidth()
                    .weight(1f - canvasHeight, true)
```

```
                            .zIndex(2f)
                    )
                }
            }
        }
    }
}
```

kotlin/main.kt

```
import application.Application
import logic.GraphFileReader
import mu.KotlinLogging
import org.apache.log4j.BasicConfigurator
import org.apache.log4j.ConsoleAppender
import org.apache.log4j.PatternLayout

val logger = KotlinLogging.logger {}

fun main() {
    BasicConfigurator.configure(ConsoleAppender(PatternLayout("[%p]
[%d{HH:mm:ss}] %n [%l] %n %m %n%n")))
    val application = Application()
    application.start()
}
```