**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по учебной практике**

**Тема: Реализация алгоритма А\* на языке Java с графическим интерфейсом.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 0382 |  | Кривенцова Л.С. |
| Студентка гр. 0382 |  | Деткова А.С. |
| Студентка гр. 0382 |  | Здобнова К.Д. |
| Руководитель |  | Голубева В.П. |

Санкт-Петербург

2022

**ЗАДАНИЕ**

**НА УЧЕБНУЮ ПРАКТИКУ**

|  |  |  |
| --- | --- | --- |
| Студентка Кривенцова Л.С. группы 0382 | | |
| Студентка Деткова А.С. группы 0382 | | |
| Студентка Здобнова К.Д. группы 0382  Тема практики: Реализация алгоритма А\* на языке Java с графическим интерфейсом. | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: А\*. | | |
| Сроки прохождения практики: 29.06.2022 – 12.07.2022 | | |
| Дата сдачи отчета: 12.07.2022 | | |
| Дата защиты отчета: 12.07.2022 | | |
|  | | |
| Студентка гр. 0382 |  | Кривенцова Л.С. |
| Студентка гр. 0382 |  | Деткова А.С. |
| Студентка гр. 0382 |  | Здобнова К.Д. |
| Руководитель |  | Голубева В.П. |

**АННОТАЦИЯ**

Основной целью работы является получение навыков программирования на языке Java и освоение парадигмы объектно-ориентированного программирования. Цель достигается командной работой путём разработки программы с графическим интерфейсом, использующей указанные в задании алгоритмы. В данной работе необходимо реализовать алгоритм А\*, находящий во взвешенном графе маршрут наименьшей стоимости от выбранной начальной вершины до выбранной конечной.

**SUMMARY**

The main purpose of the work is to gain programming skills in the Java language and master the paradigm of object-oriented programming. The goal is achieved by teamwork by developing a program with a graphical interface that uses the algorithms specified in the task. In this work, it is necessary to implement the A\* algorithm, which finds the least cost route from the selected initial vertex to the selected final vertex in the weighted graph.

**СОДЕРЖАНИЕ**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Требования к программе | 6 |
| 1.1. | Исходные требования к программе\*   * + 1. Общие исходные Требования     2. Требования к вводу исходных данных | 0 |
| 1.2. | Уточнение требований после сдачи прототипа | 0 |
| 1.3.  1.4 | Уточнение требований после сдачи 1-ой версии  Уточнение требований после сдачи 2-ой версии | 0  0 |
| 2. | План разработки и распределение ролей в бригаде | 0 |
| 2.1. | План разработки | 0 |
| 2.2. | Распределение ролей в бригаде | 0 |
| 3. | Особенности реализации | 0 |
| 3.1. | Возможности приложения | 0 |
| 3.2. | Структуры данных | 0 |
| 3.3 | Основные методы | 0 |
| 4. | Тестирование | 0 |
| 4.1 | Тестирование графического интерфейса | 0 |
| 4.2 | Тестирование кода алгоритма | 0 |
| 4.3 | … | 0 |
|  | Заключение | 0 |
|  | Список использованных источников | 0 |
|  | Приложение А. Исходный код – только в электронном виде | 0 |

**ВВЕДЕНИЕ**

Задача практики состоит в разработке приложения, позволяющего находить минимальный маршрут в взвешенном графе. Программа визуализирует пошаговое нахождение такого пути на графе. Для его поиска используется алгоритм А\*.

Поиск A\* (произносится «А звезда» или «А стар», от англ.Astar)—в информатике и математике, алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной).

Порядок обхода вершин определяется эвристической функцией «расстояние + стоимость» (обычно обозначаемой как f(x)). Эта функция—сумма двух других: функции стоимости достижения рассматриваемой вершины (x) из начальной (обычно обозначается как g(x) и может быть как эвристической, так и нет), и функции эвристической оценки расстояния от рассматриваемой вершины к конечной (обозначается как h(x)).

Функция h(x) должна быть допустимой эвристической оценкой, то есть не должна переоценивать расстояния к целевой вершине. Например, для задачи маршрутизации h(x) может представлять собой расстояние до цели по прямой линии, так как это физически наименьшее возможное расстояние между двумя точками.

**1. ТРЕБОВАНИЯ К ПРОГРАММЕ**

**1.1. Исходные Требования к программе**

**1.1.1 Общие исходные Требования:**

- Приложение должно быть с графическим интерфейсом.

- Приложение должно быть ясным и удобным для пользователя.

- Помимо визуализации алгоритма, должны выводиться текстовые пояснения происходящего для пользователя.

- Визуализация алгоритма должна быть пошаговой, шаги не должны быть крупными.

- Должна быть возможность задать входные данные как из файла, так и при работе в самом приложении.

- Следует предусмотреть возможности взаимодействия с графическими элементами с помощью мыши (простейший пример: создание вершины графа в заданном месте на холсте по щелчку).

Макет приложения:

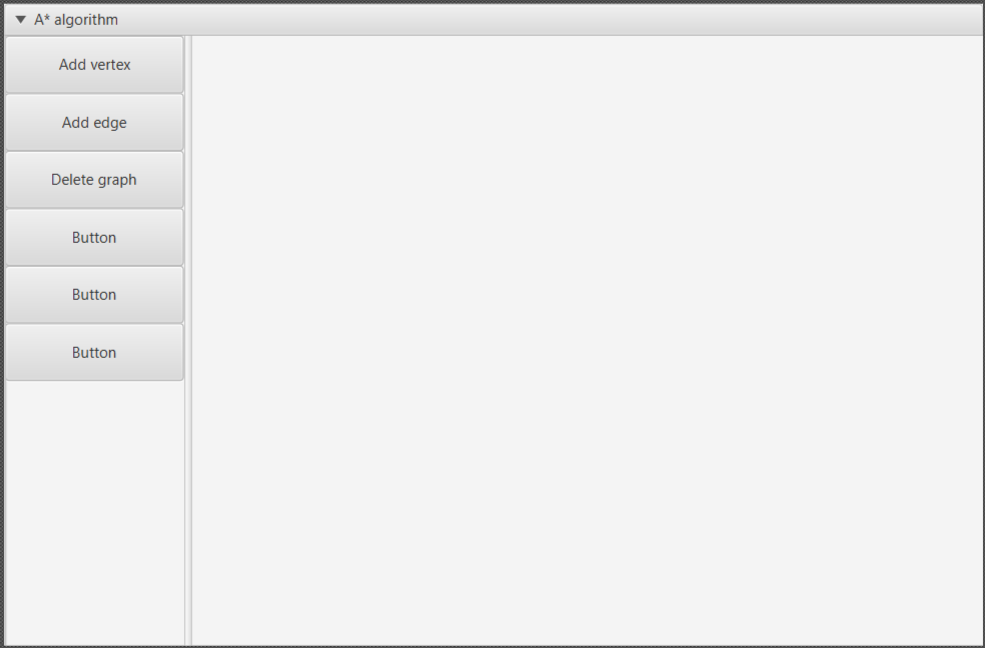


Рисунок 1. Макет приложения

**1.1.3 Требования к вводу исходных данных:**

Подраздел «Исходные Требования к программе» следует разбить на подразделы 2-го уровня (1.1.1 – требования к вводу исходных данных, 1.1.2 – требования к визуализации и т.д.)

**1.2. Уточнение требований после сдачи прототипа**

1) Упростить добавление вершины: вместо кнопки сделать добавление кликом мыши по полю. При появлении вершины задать ей дефолтное имя.

2) Добавить перетаскивание вершин.

3) Добавить кнопки пошагового выполнения работы алгоритма А\*.

4) Добавить запуск приложения через консоль.

**2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ**

**2.1. План разработки**

1. Завершение прохождения курса на платформе Stepik – **30 июня**.

2. Создание общего репозитория. Выбор графического интерфейса и создание прототипа для GUI. Сдача вводного задания и согласование спецификации и плана разработки. Написание класса для реализации алгоритма А\*. – **1 июля.**

3. Создание прототипа - приложение, демонстрирующее интерфейс, но (почти) не реализующее основные функции. Прототип включает в себя: стартовое окно приложения, окно редактора графа; создание кнопок для редактирования, поля для ввода данных; написание классов для хранения и обработки графа; дизайн интерфейса – **4 июля**.

4. Добавление возможности чтения данных из файла, написание методов для отрисовки редактирования графа. Добавление возможности запустить алгоритм А\* для введенного графа. Сдача первой версии программы – **8 июля**.

5. Добавление тестирования и исключений для крайних случаев. Добавление пошаговой визуализации алгоритма А\*. Сдача второй версии программы – **11 июля**.

6. Завершить разработку проекта – **12 июля.**

**2.2. Распределение ролей в бригаде**

1. Разработка прототипа:

Кривенцова Любовь – написание алгоритма А\*.

Деткова Анна – написание классов для работы с графом и его отрисовки.

Здобнова Ксения – написание отчета, создание макета графического интерфейса и его дизайн, написание классов для работы с графом и его отрисовки.

2. Разработка 1-ой версии программы:

Кривенцова Любовь – написание тестов для алгоритма А\*. Добавление алгоритма А\* в приложение. Написание отчета.

Деткова Анна – добавление в приложение функции для запуска алгоритма А\*. Визуализация алгоритма в приложении. Добавление редактирования графа с помощью нажатия мыши.

Здобнова Ксения – визуализация в приложении методов редактирования графа. Визуализация алгоритма в приложении. Добавление редактирования графа с помощью нажатия мыши. Написание отчета.

3. Разработка 2-ой версии программы:

Кривенцова Любовь – написание Исключений для алгоритма А\* для ввода данных как через файл, так и через специального окна приложения. Проверка работы алгоритма на крайних случаях. Написание отчета.

Деткова Анна – добавление исключений для методов редактирования графа. Добавление пошаговой визуализации работы с графом. Написание отчета.

Здобнова Ксения – добавление исключений, возникающих при взаимодействии пользователя с инструментами редактирования графа. Добавление пошаговой визуализации работы с графом. Написание отчета.

**3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ**

**3.1 Возможности приложения**

Приложение запускается со стартового окна, функционал работы с графом предоставляется после нажатия на кнопку «*Start*»:

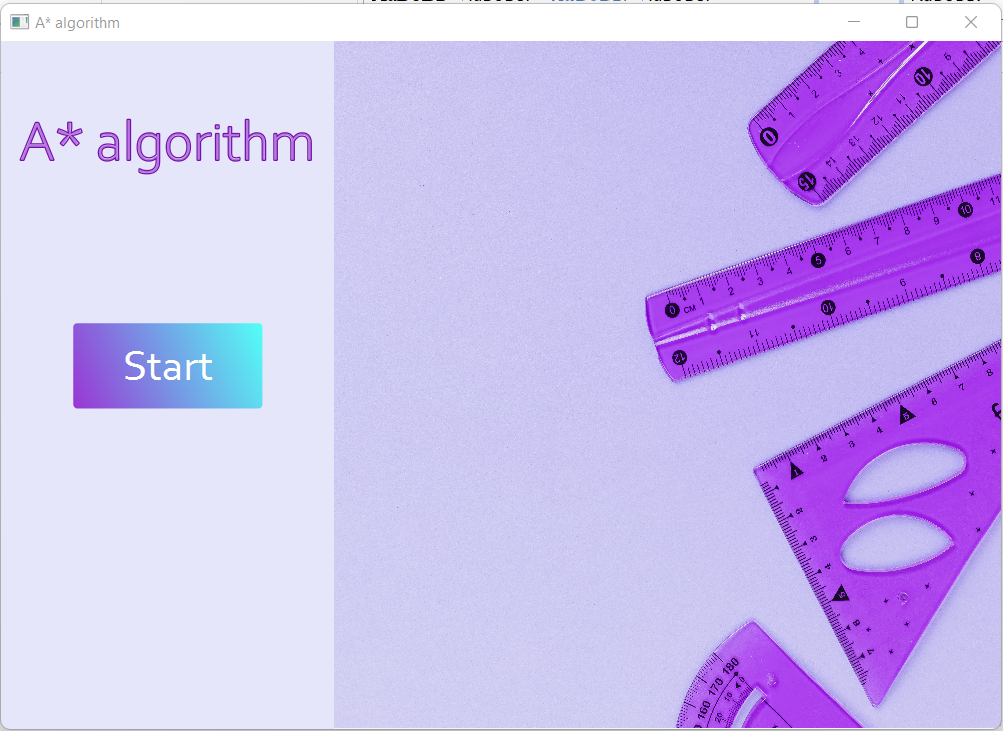


Рисунок 1. Стартовое окно

Далее пользователю открывается редактор, в котором можно работать с графом.

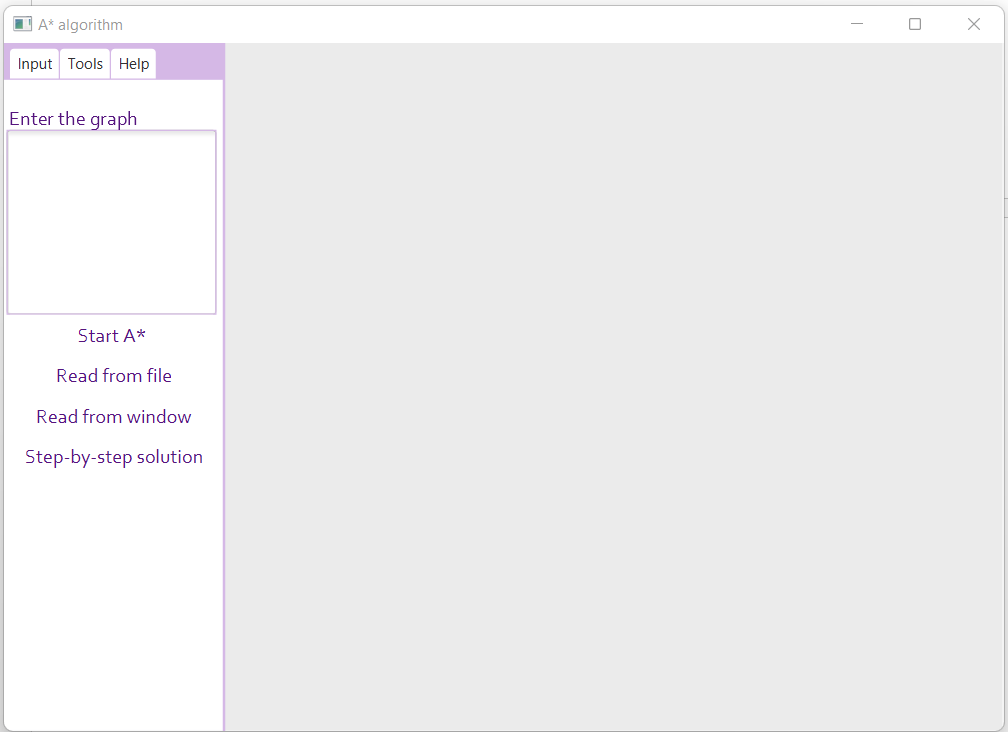


Рисунок 2. Окно редактора

Во вкладке «*Input*» пользователь может ввести граф для последующей работы с ним. Кнопка «*Start A\**» запускает алгоритм А\*. «*Read from file*» дает возможность пользователю считать данные о графе из файла – название файла вбивается пользователем в специальном окне, которое появляется после нажатия соответствующей кнопки. «*Read from window*» запускает алгоритм по данным, введенным в окно, расположенном на панели. «*Step-by-step-solution*» запускает пошаговое выполнение алгоритма на графе – пользователь может идти пошагово посмотреть работу алгоритма, будут доступны кнопки возвращения на шаг назад и на шаг вперед.

Во вкладке «*Tools*» расположены инструменты редактирования графа. Кнопка «Add edge» добавляет ребро - пользователь кликает мышкой по двум вершинам, между которыми нужно провести ребро, затем открывается окно для ввода веса ребра. «Delete vertex» удаляет вершину – пользователь кликает по вершине, которую следует удалить. «Delete edge» удаляет ребро – пользователь кликает мышкой по двум вершинам, между которыми нужно удалить ребро. «Clear» - очищает поле.

Во вкладке «*Help*» описаны требования к формату ввода данных и вывод.

Двойным кликом правой кнопки мыши по полю пользователь может добавить вершину, при добавлении ей задается дефолтное имя. При одинарном нажатии на вершину правой кнопки мыши открывается контекстное меню с двумя доступными пользователю функциями – переименовать вершину и удалить вершину.

**3.2. Структуры данных**

1) Алгоритм А\*.

Разработайте программу, которая решает задачу построения кратчайшего пути в ориентированном графе методом А\*. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес.

Для работы алгоритма был написан класс AStar, имеющий следующую структуру:

public class AStar {

private HashMap<Vertex, Double> f; - хэш-таблица, соотносящая вершину и её сумму пройденного расстояния и эвристического веса.

private HashMap<Vertex, Double> g; - хэш-таблица, соотносящая вершину и её сумму пройденного расстояния.

private ArrayList<Vertex> in\_open; - массив, реализующий очередь вершин, из которой в алгоритме вершины берутся для рассмотрения.

private ArrayList<Vertex> in\_closed; - массив пройденных (рассмотренных) вершин.

private HashMap<Vertex, Vertex> from; - хэш-таблица, где ключом является вершина, а значением – вершина, путём из которой мы пришли в вершину, указанную в ключе.

private List<ArrayList<Vertex>> open; - двумерный массив, содержащий массивы состояний очереди на каждом шагу.

private List<ArrayList<Vertex>> closed; - двумерный массив, содержащий массивы рассмотренных вершин на каждом шагу.

private ArrayList<Vertex> solution; - массив вершин, состовляющий итоговый, найденный в алгоритме, кратчайший путь.

private String path = ""; - строка, в которую записывается итоговый путь.

public AStar();- конструктор, инициализирует поля пустыми списками и хеш-таблицами.

public static double h(Vertex a, Vertex b); — эвристика двух вершин — рассчитывает евклидово расстояние. Принимает на вход две вершины, между которыми рассчитывается расстояние, и возвращает вещественное число – рассчитанное евклидово расстояние.

public Vertex min\_f(); - метод не принимает ничего в качестве аргументов, возвращает наиболее подходящую вершину, куда следует сделать следующий шаг алгоритма (выбирает из очереди вершину, с наименьшим показателем f).

private Vertex a\_star(Vertex start, Vertex finish, ArrayList<Vertex> graph); — алгоритм А\*, в качестве аргументов принимает начальную и конечную вершины (ссылки) и массив вершин, представляющий собой граф; возвращает finish вершину, или null, если путь построить не удалось.

public ArrayList<Vertex> a\_star\_public(Vertex start, Vertex finish, ArrayList<Vertex> graph); - «разворачивает» цепочку вершин. Продвигаясь по полям from проходит от конечной вершины (полученной в методе a\_star) до начальной, и восстанавливает путь, который возвращает как список вершин — кратчайший путь от start до finish.

public String getPath(); - метод, проходясь по вершинам пути, заполняет поле-строку path и возвращает значения поля.

public double getWeight(); - метод проходится по вершинам итогового пути (solution) и считает суммарный вес, который возвращает в форме вещественного числа.

public void SetF(HashMap<Vertex, Double> new\_f); - метод позволяет установить зависимости вершин и их f-значений. Используется для проведения тестов (чтобы задать данные для тестирования).

public void SetInOpen(ArrayList<Vertex> new\_open); - метод позволяет установить очередь вершин. Используется для проведения тестов (для того чтобы задать данные для тестирования).

public List<ArrayList<Vertex>> getOpen(); public List<ArrayList<Vertex>> getClose();- метод для получения в других классах значения полей open и closed(получить состояние очереди и закрытых вершин на каждом шагу. Используется для демонстрирования решения в режиме step-by-step)

Подробнее про алгоритм А\* и реализующий его метод a\_star.

В очередь заносится стартовая вершина, её расстояние от начала равно нулю, а вес высчитывается как значение эвристической функции.

Пока в очереди есть вершины, продолжается цикл while (с условием, пока очередь не пуста). Если очередь оказывается пустой, значит пути нет и функция, не заходя в цикл, возвращает значение null.

В цикле выбирается текущая вершина, методом определения вершины из с наименьшим весом (находит такую вершину метод min\_f). Так, в current хранится текущая вершина.

Так как логика алгоритма предполагает, что начальная и конечная вершины не совпадают, это отражено в функционале приложения (нельзя выбрать одну вершину и в качестве начала, и в качестве конца). Таким образом, обрабатывать такой случай не требуется.

Такая вершина удаляется из очереди и заносится в массив пройденных вершин. Соответственно меняются поля in\_open и in\_closed.

Запускается подцикл for, проходящий по соседям текущей вершины. Для рассматриваемой соседней вершины рассчитывается (но не записывается в данные поля g) расстояние от начала до неё. Если вершина не находится в очереди, или рассчитанное расстояние меньше расстояния, записанного в данных поля g, то заполняются соответствующие зависимости соседа в полях from, f, g. Если рассматриваемая соседняя вершина не находится в очереди (и она не пройдена), встаёт в очередь. Итерация цикла завершается.

2) Coordinates <T> - класс координат. Принимает любой класс и хранит объекты переданного типа. Необходим для хранения координат вершины.

Поля: private T x, private T y — координата x и y соответственно, имеют тип T (передается при создании объекта.

Методы:

Coordinates (T x, T y) — конструктор, заполняет поля переданными аргументами.

public T getX() - возвращает координату x.

public T getY() - возвращает координату y.

public void setX(T x) — устанавливает новое значение координаты х.

public void setY(T y) — устанавливает новое значение координаты y.

3) Vertex — класс вершины. Хранит символьное имя вершины, ее координаты и ее соседей, т. е. ребра выходящие из этой вершины с весами.

Поля:

char name — имя вершины, Coordinates<Double> coordinates — координаты по x и y типа Double, HashMap<Vertex, Double> neighbours — хеш-таблица для соседей вершины (ребер исходящих из вершины), ключ — другая вершина, тип Vertex, значение — вес ребра типа Double.

Методы:

public Vertex(char name, double x, double y) — конструктор, заполняет поля переданными значениями, задает имя и координаты вершины, создает новый объект для хранения соседей вершины.

void addNeighbour(Vertex vertex, double distance) — добавляет нового соседа вершины, дополняет хеш-таблицу neighbours новым элементом.

void deleteNeighbour(Vertex vertex) — удаляет вершину-соседа.

public char getName() - возвращает символьное имя вершины.

HashMap<Vertex, Double> getNeighbours() - возвращает хеш-таблицу соседних вершин.

public Coordinates<Double> getCoordinates() - возвращает координаты вершины.

public void setCoordinates(Double x, Double y) — устанавливает новые, переданные в качестве аргумента, координаты.

4) Graph — класс графа. Хранит список вершин, а также хеш-таблицу допустимых значений имен для вершин.

Поля:

private ArrayList<Vertex> vertexes — список вершин, входящих в граф.

private HashMap<Character, Boolean> availableNames — хеш-таблица допустимых имен вершин графа. Ключ — допустимое имя, значение — булева переменная, если true имя свободно, иначе — занято. Таким образом, все имена вершин графа разные.

Методы:

public ArrayList<Vertex> getVertexes() - возвращает список вершин графа.

public Graph() - конструктор, создает пустой список вершин графа и заполняет хеш-таблицу допустимых имен символами в ключах и true в значениях.

public Character getAvailableName() - находит и возвращает первое доступное имя для вершины.

void addVertex(char vertex, double x, double y) — добавляет вершину в список вершин графа, создается новый объект класса Vertex и добавляется в список vertexes.

void addVertex(Vertex vertex) — добавляет уже созданный объект класса Vertex.

void addEdge(Vertex first, Vertex second, double weight) — добавляет ребро из first в second с весом weight в хеш-таблицу соседей вершины first.

void deleteVertex(char vertex) — удаляет вершину по имени из списка вершин, а также из всех хеш-таблиц вершин, которые соседствовали с переданной.

void deleteVertex(Vertex vertex) — аналогично предыдущему методу, но передается объект класса Vertex.

void deleteEdge(char first, char second) — удаляет ребро, ведущее из вершины с именем first в вершину с именем second.

void deleteEdge(Vertex first, Vertex second) — анлогично предыдущему методу удаляет ребро между двумя вершинами, но по объектам типа Vertex, а не именам вершин.

Vertex findVertex(char name) — находит вершину с переданным именем, если такой не найдено, то возвращает null.

void clearGraph() - очищает граф. Очищает список вершин.

5) VertexDravable — граф графического отображения вершины графа.

Поля:

private char name — имя вершины.

private Circle vertexCircle — объект типа Circle из библиотеки javaFX, круг, представляет собой вершину графа в графическом приложении.

private Text nameCircle — объект типа Text из библиотеки javaFX, выводит имя вершины.

private Vertex vertex — вершина, чье графическое представление хранится в объекте.

private Coordinates coordinates — координаты вершины.

private final static double radiusOfVertexCircle = 18.0 — радиус круга для вершины, статическая константа.

Методы:

public VertexDrawable(Vertex vertex) — конструктор.

public void setName(String name) — изменить имя вершины.

public Circle getView() - возвращает графический объект графа, кружок.

public Text getName() - возвращает графический объект с именем вершины, значок с именем вершины.

public Vertex getVertex() - возвращает вершину, соответствующую отрисовываемой вершине.

public void moveCircle() - меняет координаты вершины и ее графического отображения при передвижении вершины мышкой.

6) EdgeDrawable — класс для графического отображения ребра графа между двумя вершинами. Для двунаправленного ребра такой объект создается один.

Поля:

private double weightFromFirstToSecond, weightFromSecondToFirst — веса в одну и другую сторону.

private Vertex first, second — первая и вторая вершина ребра.

private boolean directionFromFirstToSecond, directionFromSecondToFirst — булевые переменные, показывают направление ребра, от первой вершине ко второй, наоборот, или сразу в обе стороны.

private Line edgeLine — объект типа Line из библиотеки javaFX, линия, представляет собой ребро графа.

private Text weightView — объект типа Text из библиотеки javaFX, выводит вес ребра.

private final static double radiusOfVertexCircle = 18.0 — радиус круга, изображащего вершину графа, статическая константа.

Методы:

public EdgeDrawable(Vertex first, Vertex second, double weight) — конструктор, задает координаты, заполняет поля.

public void setReversedDirection(double weight) — устанавливает направление ребра в обратную сторону.

public void setReversedDirection(Vertex start, Vertex finish, double weight) — устанавливает направление ребра в обратную сторону для двух конкретных вершин.

public void deleteOneWay(Vertex first, Vertex second) — удаляет один путь в одном направлении.

public boolean isTwoDirectional() - проверяет является ли ребро двунаправленным.

public boolean isOnlyOneWay() - проверят является ли ребро однонаправленным.

public boolean isReverse(Vertex first, Vertex second) — проверяет является ли ребро инверсированным относительно введенных вершин.

public boolean cmpFirst(Vertex other) — сравнивает поданную вершину и первую ребра.

public boolean cmpSecond(Vertex other) — сравнивает поданную вершину и вторую вершину ребра.

public boolean cmpBoth(Vertex first, Vertex second) — сравнивает обе вершины ребра и первую, и вторую с введенными, а также на реверсивность.

public Text getName() — озвращает графический объект с весом вершины, значок с весом ребра в формате: а→b: 10| b→a: 20.3.

public Line getView() — возвращает графический объект графа, линию — ребро.

public Vertex getFirst(), public Vertex getSecond() - возвращает соответствующие вершины из полей ребра.

public void moveLine() - меняет координаты линии — графического отображения ребра, и его веса.

7) GraphController — класс отвечающих за всю логику связанную с графом. Связывает отрисовку с самим графом. Поиск вершин, ребер, запуск алгоритма А\*, считывание графа, установка реакций на события у вешин (кружков, их изображений), отрисовка графа, обертка над удалением вершины, ребра, добавлением вершины, ребра.

Поля:

ArrayList<VertexDrawable> vertexesDrawable — список отрисовываемых вершин.

Graph graph — граф.

ArrayList<EdgeDrawable> edgesDrawable — список отрисовываемых ребер.

AStar aStar — объект для А\*.

Vertex chosen1, chosen2 — 2 выбранные вершины на графе, нужны для удаления ребра между двумя вершинами, добавления ребра между двумя вершинами, запуска А\* между двумя вершинами.

Методы:

public GraphController() - инициализация полей объекта класса, создание объекта класса Graph, создание пустых списков всех отрисовываемых объектов.

public void drawVertex(Pane pane, Vertex vertex) — создание и добавление новой вершины в граф и в список отрисовок вершин.

private VertexDrawable findVertex(Vertex vertex) — поиск отрисовки вершины по объекту Vertex.

private EdgeDrawable findEdge(Vertex start, Vertex finish) — находит ребро между двумя вершинами.

public void runningAlgorithmAStar(Vertex first, Vertex second) — запуск алгоритма А\*, для этого пользователь должен выделить две вершины двойным кликом мышки, первая — начало, вторая — конец. Отрисовывает результат работы алгоритма, путь от начала до конца подсвечивается зеленым (вершины и ребра).

public void readGraphFromWindow(String string) — считывает граф в нужном формате из строки, переданной функцию. Заполняет граф и его графику.

public void readGraphFromFile(String fileName) — считывает граф из файла, имя которого передается в функцию.

protected void setEventHandlers() - устанавливает обработчики событий для вершин графа, при клике, удерживании или перетаскиваниями с вершинами будет происходить преобразования.

public void drawGraph(Pane pane) — отрисовывает граф.

public void addEdge(double weight) — добавление ребра между двух выбранных вершин chosen1 и chosen2.

public void deleteEdge() - удаление ребра между двумя вершинами.

public void deleteVertex(VertexDrawable vertexDrawable) — удаляет вершину, которую выбрал пользователь (передается в метод).

**3.3. Основные методы**

**4. ТЕСТИРОВАНИЕ**

**4.1. Первый подраздел третьего раздела**

**4.2. Второй подраздел третьего раздела**

**ЗАКЛЮЧЕНИЕ**

Кратко подвести итоги, проанализировать соответствие поставленной цели и полученного результата.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

***Ниже представлены примеры библиографического описания, В КАЧЕСТВЕ НАЗВАНИЯ ИСТОЧНИКА в примерах приводится вариант, в котором применяется то или иное библиографическое описание.***

1. Иванов И. И. Книга одного-трех авторов. М.: Издательство, 2010. 000 с.

2. Книга четырех авторов / И. И. Иванов, П. П. Петров, С. С. Сидоров, В. В. Васильев. СПб.: Издательство, 2010. 000 с.

3. Книга пяти и более авторов / И. И. Иванов, П. П. Петров, С. С. Сидоров и др.. СПб.: Издательство, 2010. 000 с.

4. Описание книги под редакцией / под ред. И.И. Иванова СПб., Издательство, 2010. 000 с.

5. Иванов И.И. Описание учебного пособия и текста лекций: учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2010. 000 с.

6. Описание методических указаний / сост.: И.И. Иванов, П.П. Петров. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2010. 000 с.

7. Иванов И.И. Описание статьи с одним-тремя авторами из журнала // Название журнала. 2010, вып. (№) 00. С. 000–000.

8. Описание статьи с четырьмя и более авторами из журнала / И. И. Иванов, П. П. Петров, С. С. Сидоров и др. // Название журнала. 2010, вып. (№) 00. С. 000–000.

9. Иванов И.И. Описание тезисов доклада с одним-тремя авторами / Название конференции: тез. докл. III международной науч.-техн. конф., СПб, 00–00 янв. 2000 г. / СПбГЭТУ «ЛЭТИ», СПБ, 2010, С. 000–000.

10. Описание тезисов доклада с четырьмя и более авторами / И. И. Иванов, П. П. Петров, С. С. Сидоров и др. // Название конференции: тез. докл. III международной науч.-техн. конф., СПб, 00–00 янв. 2000 г. / СПбГЭТУ «ЛЭТИ», СПБ, 2010, С. 000–000.

11. Описание электронного ресурса // Наименование сайта. URL: http://east-front.narod.ru/memo/latchford.htm (дата обращения: 00.00.2010).

12. ГОСТ 0.0–00. Описание стандартов. М.: Изд-во стандартов, 2010.

13. Пат. RU 00000000. Описание патентных документов / И. И. Иванов, П. П. Петров, С. С. Сидоров. Опубл. 00.00.2010. Бюл. № 00.

14. Иванов И.И. Описание авторефератов диссертаций: автореф. дисс. канд. техн. наук / СПбГЭТУ «ЛЭТИ», СПБ, 2010.

15. Описание федерального закона: Федер. закон [принят Гос. Думой 00.00.2010] // Собрание законодательств РФ. 2010. № 00. Ст. 00. С. 000–000.

16. Описание федерального постановления: постановление Правительства Рос. Федерации от 00.00.2010 № 00000 // Опубликовавшее издание. 2010. № 0. С. 000–000.

17. Описание указа: указ Президента РФ от 00.00.2010 № 00 // Опубликовавшее издание. 2010. № 0. С. 000–000.

**ПРИЛОЖЕНИЕ А**

**НАЗВАНИЕ ПРИЛОЖЕНИЯ**

полный код программы должен быть в приложении, печатать его не надо