

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

**ОТЧЕТ**  
**по учебной практике**  
**Тема: Алгоритм А\***

Студент гр. 9381

Любимов В.А.

Студент гр. 9381

Щеглов Д.А.

Студент гр. 9381

Гурин С.Н.

Руководитель

Ефремов М.А

Санкт-Петербург

2021

## ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Любимов В.А., гр. 9381

Студент Щеглов Д.А., гр. 9381

Студент Гурин С.Н., гр. 9381

Тема практики: Алгоритм А\*

Задание на практику:

Разработать визуализатор алгоритма А\* на языке Java

Сроки прохождения практики: 01.07.2021 – 14.07.2021

Дата сдачи отчета: 14.07.2021

Дата защиты отчета:

Студент

Любимов В.А.

Студент

Щеглов Д.А.

Студент

Гурин С.Н.

Руководитель

Ефремов М.А.

## **АННОТАЦИЯ**

Данная работа предусматривает создание GUI-приложения, позволяющее создать граф, а так же визуализировать на этом графе работу алгоритма A\*.

Разработка данного приложения происходит на языке Java командой из 3-х человек. Роли каждого человека распределены в соответствии с

## **SUMMARY**

This practical work provides for the creation of a GUI application that allows you to create a graph, as well as visualize the work of the A\*algorithm on this graph.

The development of this application takes place in the Java language by a team of 3 people. The roles of each person are distributed in accordance with these tasks.

## СОДЕРЖАНИЕ

	Введение	5
1.	Спецификация программы и организация работы	6
1.1.	Диаграммы Use-Case и Классов	6
1.2.	Организация работы: план разработки и распределение обязанностей	7
2.	Описание алгоритма	8
2.1.	Алгоритм A*	8
2.2.	Реализация алгоритма	9
2.3.	Реализации графа	10
3.	Тестирование	11
	Заключение	?
	Список использованных источников	?
	Приложение А. Название приложения	?

## **ВВЕДЕНИЕ**

Целью данной практической работы является разработка GUI-приложения на языке Java. Данное приложение производит визуализацию работы алгоритма A\*.

При разработке приложения планируется реализовать графический интерфейс для создания графа. Так же важной частью является реализация пошаговой визуализации работы алгоритма A\*.

# 1. СПЕЦИФИКАЦИЯ ПРОГРАММЫ И ОРГАНИЗАЦИЯ РАБОТЫ

## 1.1. Диаграммы Use-Case и Классов

Диаграмма Use-Case:

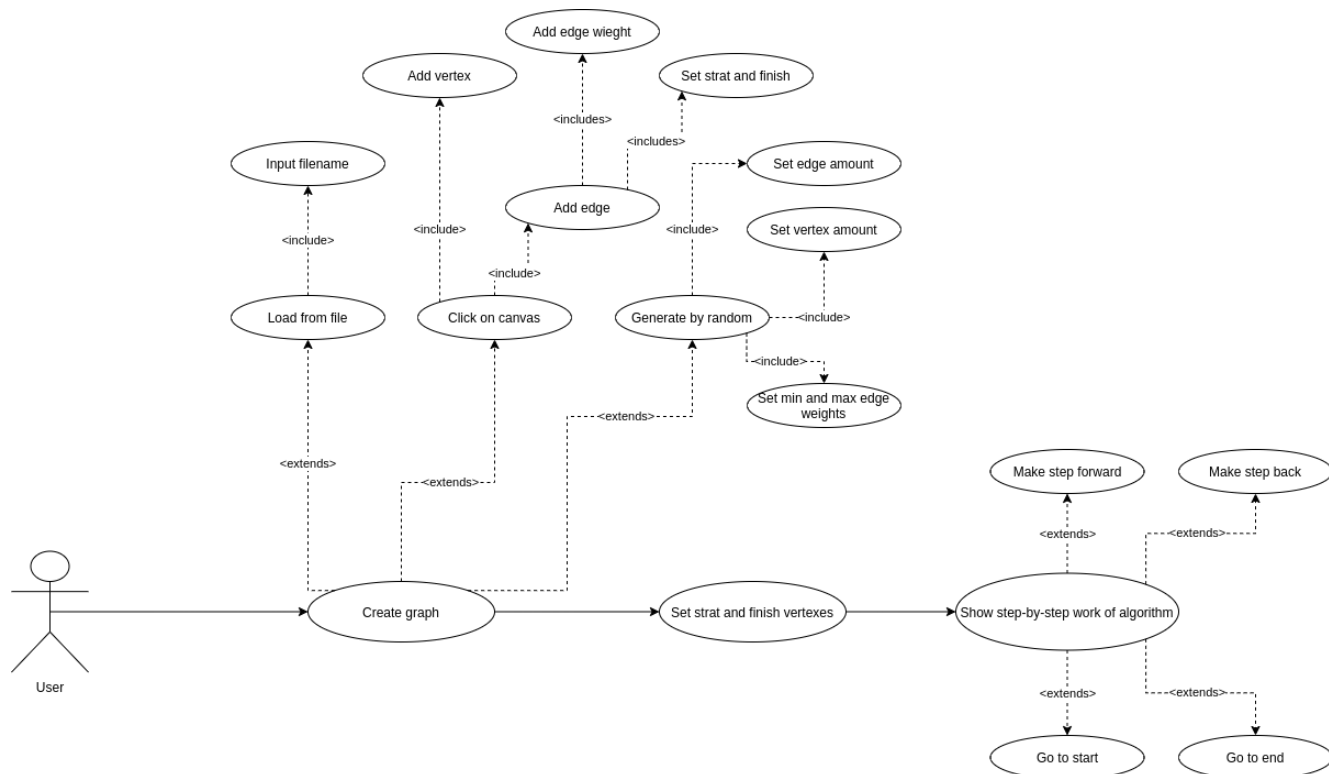


Рис. 1 Диаграмма Use-Case

## Диаграмма Классов:

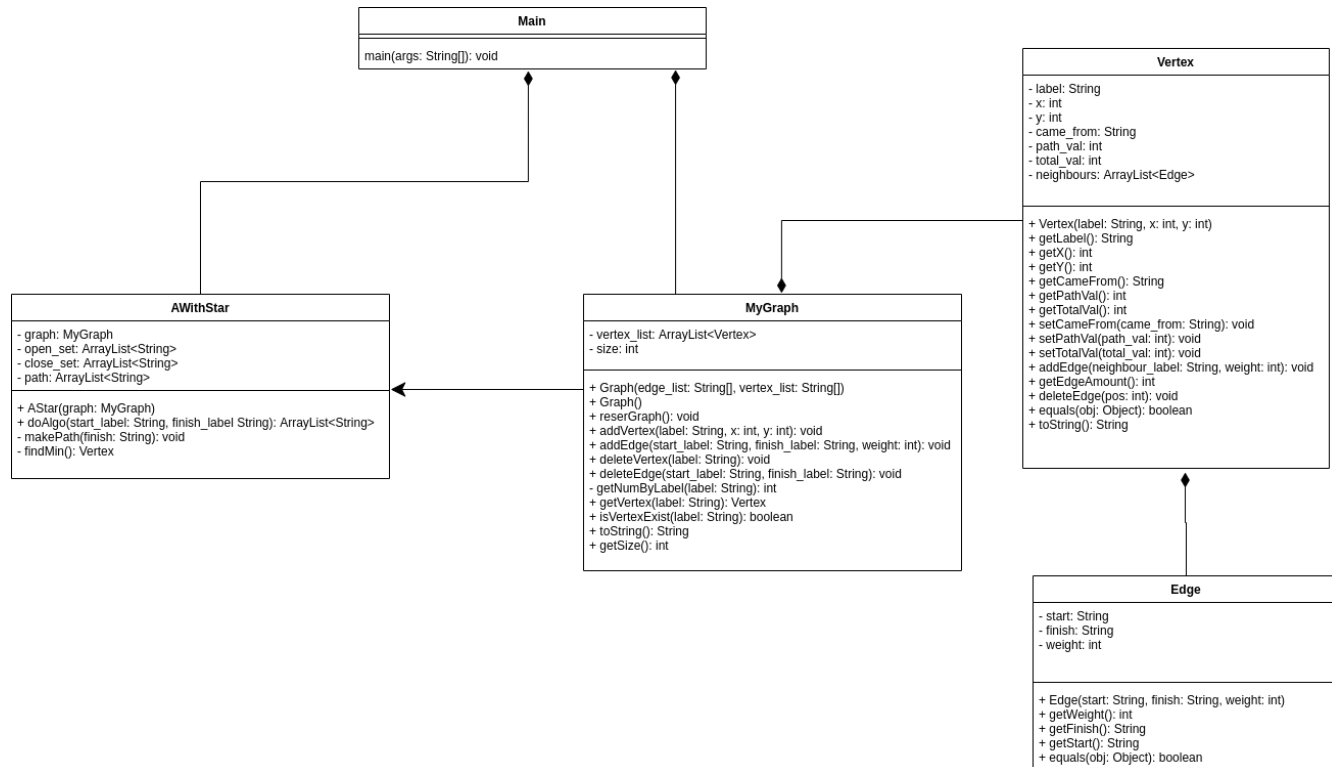


Рис. 2 Диаграмма Классов

### 1.2. План разработки

Задачи работы: изучение основ Java, создание приложения с графическим интерфейсом на Java (используя библиотеку Swing) для визуализации работы алгоритма, сборка (используя Maven) и тестирование программы.

План разработки:

1. 6 июля – создание прототипа, реализация алгоритма без демонстрации.
2. 8 июля – реализация демонстрации алгоритма в автоматическом режиме.
3. 10 июля – реализация демонстрации алгоритма в пошаговом режиме.
4. 12 июля – предоставление финальной версии программы, отчета и тестирования.

Распределение обязанностей:

Любимов В.А. – реализация алгоритма и классов бизнес-логики.

Щеглов Д.А. – реализация графического интерфейса, прототипа графического интерфейса и связи с бизнес-логики и интерфейса.

Гурин С.Н. – написание отчета, сборка и тестирование программы.

## 2. ОПИСАНИЕ АЛГОРИТМА

### 2.1. Алгоритм $A^*$

$A^*$  — это модификация алгоритма Дейкстры, оптимизированная для единственной конечной точки. Алгоритм Дейкстры может находить пути ко всем точкам,  $A^*$  находит путь к одной точке. Он отдаёт приоритет путям, которые ведут ближе к цели.

$A^*$  пошагово просматривает все пути, ведущие от начальной вершины в конечную, пока не найдёт минимальный. Как и все информированные алгоритмы поиска, он просматривает сначала те маршруты, которые «кажутся» ведущими к цели. От жадного алгоритма, который тоже является алгоритмом поиска по первому лучшему совпадению, его отличает то, что при выборе вершины он учитывает, помимо прочего, весь пройденный до неё путь. Составляющая  $g(x)$  — это стоимость пути от начальной вершины, а не от предыдущей, как в жадном алгоритме.

В начале работы просматриваются узлы, смежные с начальным; выбирается тот из них, который имеет минимальное значение  $f(x)$ , после чего этот узел раскрывается. На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа — множеством частных решений, — которое размещается в очереди с приоритетом. Приоритет пути определяется по значению  $f(x) = g(x) + h(x)$ . Алгоритм продолжает свою работу до тех пор, пока значение  $f(x)$  целевой вершины не окажется меньшим, чем любое значение в очереди, либо пока всё дерево не будет просмотрено. Из множества решений выбирается решение с наименьшей стоимостью.

Чем меньше эвристика  $h(x)$ , тем больше приоритет, поэтому для реализации очереди можно использовать сортирующие деревья.



## 2.2. Реализация алгоритма

Реализованный алгоритм описан в классе AWithStar в методе doAlgo, который принимает в себя уже построенный граф. Сначала создаются массивы open\_set и close\_set. Эти массивы являются списками оцененных вершин и раскрытых вершин соответственно. Затем идет считывание начала и конца пути. Далее происходит проверка, существует ли начальная вершина. Если нет, то программа выбрасывает исключение. Потом идет проверка с помощью процедуры checkNodes существует ли конечная вершина или является ли начальная вершина конечной. Это процедура возвращает строку с результатом проверки. Если строка оказывается не пустая, метод возвращает doAlgo возвращает путь. Далее происходит добавление начальной вершины в список оцененных вершин. Затем происходит запоминание координат финиша для дальнейших вычислений эвристики. Так же ставим значения 0 в вес пути и оценку (вес текущего пути + эвристика). Далее, пока список оцененных вершин не пуст, сначала происходит выполнение процедуры нахождения среди оцененных вершин вершину с минимальной оценкой. Если эта вершина является финишем, то происходит выполнение метода makePath, возвращающего путь к финишу. Сначала в этом методе происходит считывание предыдущей вершины из полученного пути. Затем добавление финиша в строку самого пути. И затем, пока строка с предыдущей вершиной является не пустой, происходит добавление этих предыдущих вершин, а так же замена строки теперь уже текущей вершины на предыдущую. Все это длится до тех пор, пока не дойдет до стартовой вершины, где предыдущей вершины нет (пустая строка). Затем полученный путь переворачивается. Далее, если текущая вершина не является финишем происходит оценка всех еще не раскрытых соседей текущей вершины с помощью метода openVertex. В этой процедуре программа производит цикл по всем соседям текущей вершины. Происходит сохранение соседа, затем происходит проверка, является ли этот сосед раскрытым. Если является, то проверяется следующий сосед. Далее происходит вычисление суммы веса ребра и веса полученного пути. Затем, если сосед не

является оцененным, то его добавляют в список оцененных вершин. Далее, если ранее вычисленная сумма меньше веса пути данного соседа, то этот сосед запоминается в путь и ему присваивается вес ранее вычисленной суммы. Далее вычисляется эвристика по формуле  $|cur\_x - finish\_x| + |cur\_y - finish\_y|$ , где  $cur\_x$ ,  $cur\_y$  – координаты данного соседа, а  $finish\_x$ ,  $finish\_y$  – координаты финиша. Затем оценкой пути этого соседа становится равной сумме веса ребра и веса полученного пути + эвристика. После выполнения процедуры раскрывающей вершины происходит добавление текущей вершины в список раскрытых вершин, а так же удаления ее из списка оцененных вершин. Если строка с путем ранее не возвращалась, тогда метод возвращает строку об отсутствии пути.

### 2.3. Реализация графа

Ребра графа являются объектами класса `Edge`. Ребро имеет направленность (две строки названия начальной и следующей вершины), а так же вес.

Класс ребра наследуется в класс `Vertex`. Объектом класса `Vertex` является вершина графа. Вершина имеет имя, координаты, имя предыдущего элемента пути, вес пути до этой вершины, суммарную оценку (вес пути + эвристика) и список ребер исходящих из данной вершины.

Данный граф состоит из вершин хранящихся в динамическом массиве и является членом объекта класса `MyGraph`.

Количество вершин зависит от максимального веса ребра, то есть  $max\_вершин = \text{округление\_вверх}(10 / max\_вес) * \text{округление\_вверх}(9 / max\_вес)$ . В свою очередь  $max\_ребер = 2 * max\_вершин - 2$ . Вес для минимального ребра  $\in [1, 9)$ , а вес для максимального ребра  $\in [2, 8]$ .

Конструкторы объекта класса `MyGraph` создают графы различных типов: пустой, единичный граф по количеству вершин, случайный граф, граф по списку вершин и списку ребер.

Для  $A^*$  в качестве следующей к рассмотрению вершины при равенстве суммарных оценок выбирается первая добавленная вершина в список вершин, которые необходимо рассмотреть.

$A^*$  выбрасывает исключение только, если в данном графе нет стартовой вершины. В остальных случаях будет выведен путь или сообщение об отсутствии пути.

### 3. ТЕСТИРОВАНИЕ

Тестирование реализовано с помощью библиотеки для модульного тестирования JUnit. В тестах будет сравниваться ожидаемое значение с текущим.

За G обозначим граф с вершинами A(0;0), B(2;0), C(0;2), D(4;0), E(2;2), F(4;3), I(8;3), J(4;5), H(2;5), K(8;6), L(4;6) и рёбрами AB(2), AC(2), BD(2), BE(2), DF(3), EH(3), FI(4), FJ(2), IK(4), JL(1), HJ(2), LK(4). Результаты тестирования приведены в Таблице 1.

Таблица 1 – Результаты тестирования

№	Входные данные	Выходные данные	Комментарий
1	A(0;0), B(2;0), C(0;2), D(2;4) AB(2), AC(2), BD(4).	Правильно созданный граф	GraphCorrectCreation(). Проверка метода создания графа на корректную работу.
2	A(0;0), B(2;0), C(0;2), D(2;4) AB(2), AC(2), BD(4).	IndexOutOfBoundsException	GraphCorrectCreation(). Проверка метода создания графа на корректную работу.
3	A(0;0), A(2;0), C(0;2), D(2;4) AB(2), AC(2), BD(4).	IndexOutOfBoundsException	doubleVertexCreation(). Проверка на создание графа с вершинами, имеющими одинаковые имена.
4	A(0;0), A(2;0), C(0;2), D(2;4) AU(2), AC(2), BD(4)	IndexOutOfBoundsException	wrongEdgeCreation(). Проверка на некорректное создание рёбер графа (граф получается несвязным).
5	Вершин: 10 Ребер: 12 Мин. Вес: 1 Макс. Вес: 2	IndexOutOfBoundsException	randomGraphCreation(). Проверка метода случайного создания графа на корректную работу.
6	Вершин: 10 Ребер: 12 Мин. Вес: 1 Макс. Вес: 2	IndexOutOfBoundsException	randomGraphTooManyEdgesCreation(). Проверка случайно созданного графа на превышение максимально допустимого количества рёбер.
7	Вершин: 100 Ребер: 12 Мин. Вес: 1 Макс. Вес: 6	IndexOutOfBoundsException	randomGraphTooManyVertexesCreation(). Проверка случайно созданного графа на превышение максимально допустимого количества вершин.
8			

9			
10			
11			
12			
13			
14			
15			
16			

## **ЗАКЛЮЧЕНИЕ**

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

**ПРИЛОЖЕНИЕ А**  
**НАЗВАНИЕ ПРИЛОЖЕНИЯ**