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

ОТЧЕТ

по учебной практике

Тема: Алгоритм A*

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 A*

Задание на практику:

Разработать пошаговый визуализатор алгоритма A* 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

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

При разработке приложения планируется реализовать графичесский 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:

-

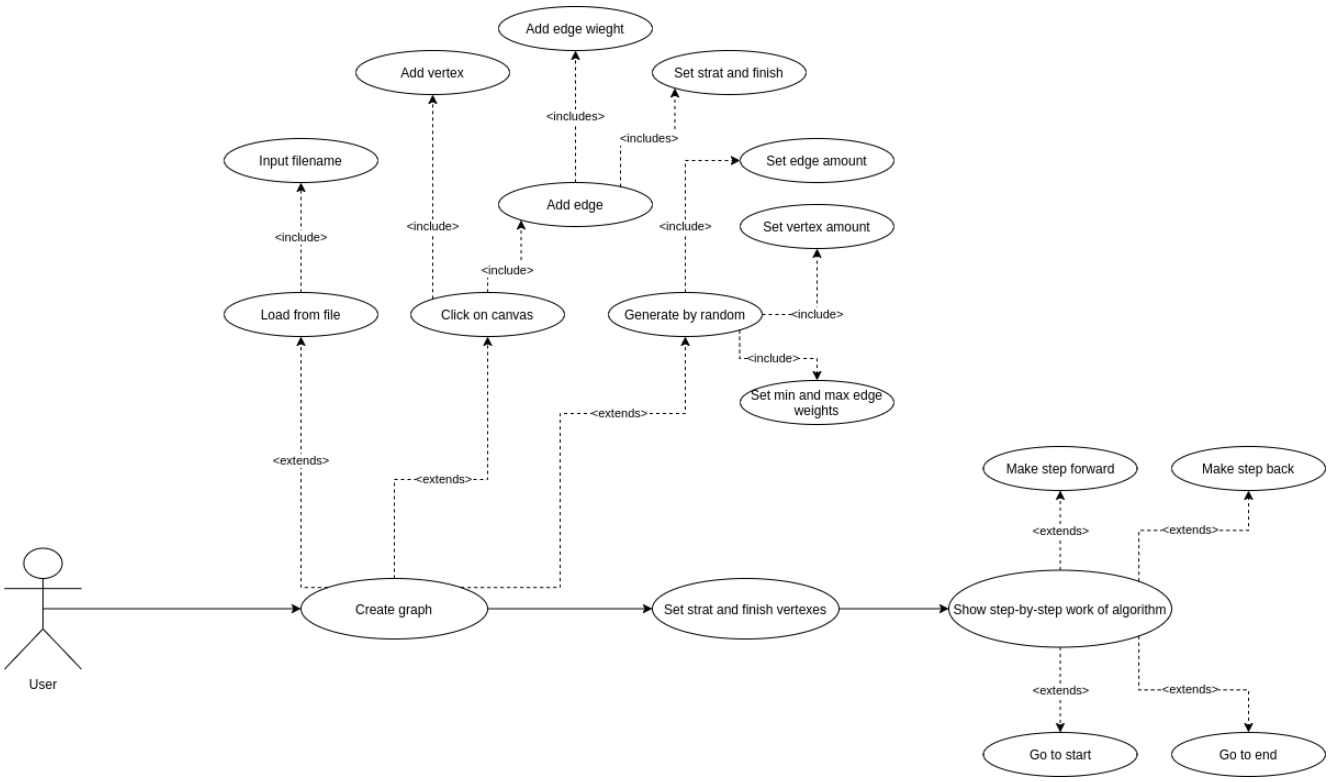

Рис. 1 Диаграмма Use-Case

При запуске программы пользователю предоставляется графический интерфейс для настройки и регулирования визуализации алгоритма A*.

Вначале пользователю необходимо создать сам граф, на котором будет продемонстрирован алгоритм. Он может сделать это тремя способами: загрузить с файла, по нажатию на поле, случайно сгенерировать. При нажатии на поле пользователь имеет возможность добавить вершину, добавить ребро, задать этому ребру вес, а так же указать направление. При случайной генерации пользователь задает колличество вершин и ребер, а так же их максимальный/минимальный вес.

Затем пользователь выбирает вершины старта и финиша.

После этого программа демонстрирует пошаговое исполнение алгоритма, имея возможность вернуться в начало, в конец, а так же имея возможность сделать шаг вперед и назад.

Диаграмма Классов:

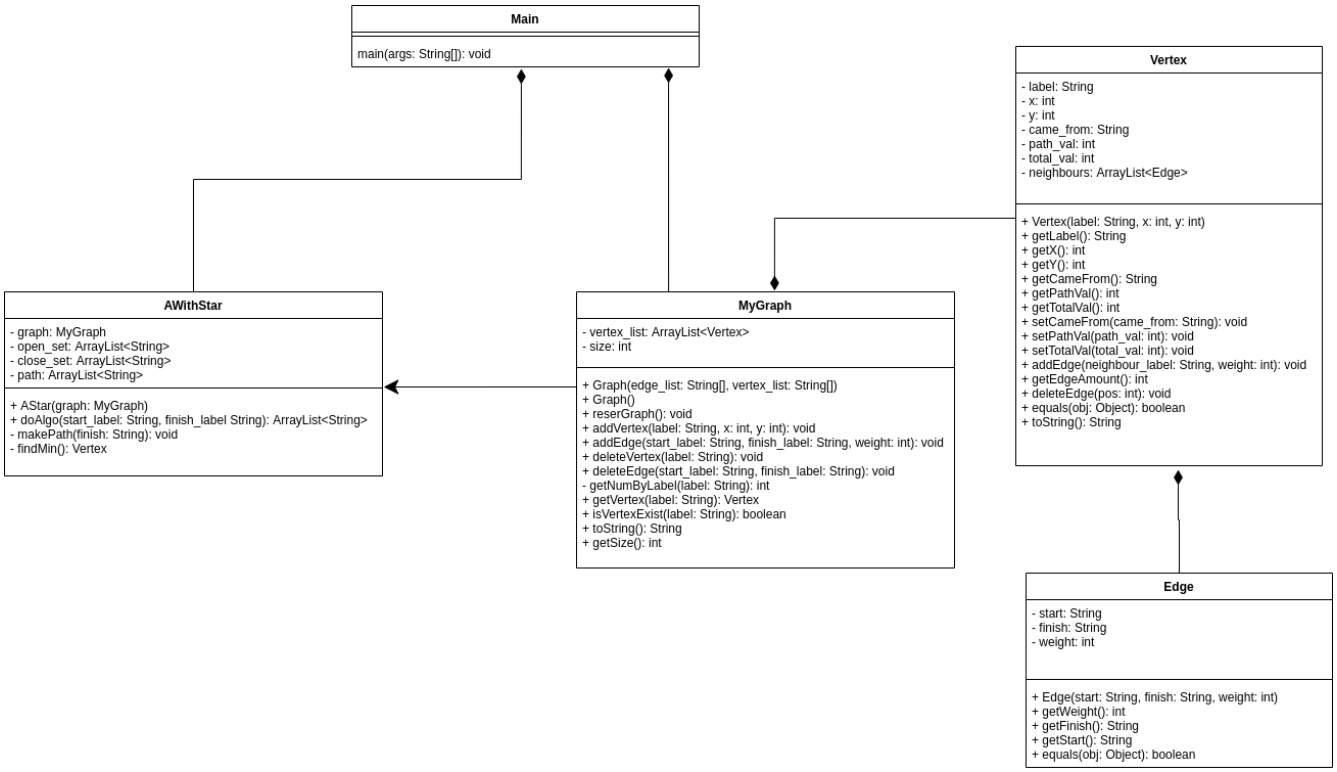

Рис. 2 Диаграмма Классов

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 интефейса.

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 раннее 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.

Колличество вершин зависит от максимального веса ребра, то есть макс_вершин = округление_вверх(10/макс_вес)*округление_вверх(9/макс_вес). В свою очередь макс_ребер = 2*макс_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 колличеству 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 превыщение максимально допустимого количества вершин.
8	Вершин: 200	IndexOutOfBoundsException	unaryGraphTooManyVertexesCreation(). Проверка графа, в котором веса всех рёбер равны 1 и все вершины связаны, на превышение максимально допустимого количества рёбер.
9	G	Количество вершин в графе	getSize(). Проверка метода, возвращающего количество вершин в графе.
10	G	Вершина добавлена	addVertex(). Проверка метода, добавляющего вершину в граф.
11	G	IndexOutOfBoundsException	addExistVertex(). Проверка на добавление в граф вершины, с уже существующим именем.
12	Вершина (0;0)	IndexOutOfBoundsException	addInSameCoordVertex(). Проверка на добавление в граф вершины с уже существующими

			коррдинатами.
13	G	Ребро добавлено	addEdge(). Проверка метода, добавляющего ребро в граф.
14	G	IOException	addEdge(). Проверка метода, добавляющего ребро в граф.
15	G	IOException	addNoStartEdge(). Проверка добавления последующих рёбер в граф.
16	G YA(2)	IOException	addNoStartEdge(). Проверка добавления последующих рёбер в граф.
17	G	Ребро добавлено	addAlreadyExistEdge(). Проверка на добавление в граф уже существующего ребра.
18	G	IOException	addAlreadyExistEdge(). Проверка на добавление в граф уже существующего ребра.
19	G AO(2)	IOException	addNoFinishEdge(). Проверка добавления в граф ребра с неправильной конечной вершиной.
20	G BA(8)	IOException	addWrongWeightEdge(). Проверка добавления в граф ребра с недопустимым весом.
21	G	Вершина A удалена	deletaVertex(). Проверка метода удаления вершины из графа.
22	G O – вершна, которую пробуем удалить, хотя	IndexOutOfBoundsException	deleteUnrealVertex(). Проверка попытки удаления из графа несуществующей вершины.

	она не входит в граф G.		
23	Пустой граф	IndexOutOfBoundsException	deleteFromEmotyVertex(). Проверка попытки удаления вершины из пустого графа.
24	G	Ребро AB(2) удалено	deleteEdge(). Проверка метода удаления ребра из графа.
25	G BA	IndexOutOfBoundsException	deleteUnrealEdge(). Проверка попытки удаления из графа несуществующего ребра.
26	G AY	IndexOutOfBoundsException	deleteWithoutFinishEdge(). Проверка попытки удалить из графа ребро с несуществующей конечной вершиной.
27	G UB	IndexOutOfBoundsException	deleteWithoutStartEdge(). Проверка попытки удалить из графа ребро с несуществующей начальной вершиной.
28	G Путь на графе: A, B, D, F, I, K.	[A, B, D, F, I, K]	doAlgoPathExist(). Проверка существования пути.
29	G Start: C Finish: A	[No path!]	doAlgoNoPath(). Проверка метода алгоритма на отсутсвие пути.
30	G Start: A Finish: A	[Start and finish vertexes are same!]	doAlgoSameStartFinish(). Проверка на совпадение начальной и конечной вершины пути.
31	G Start:A Finish: Ms	[No path! Finish vertex doesn’t exist!]	doAlgoNoFinish(). Проверка пути на корректную конечную вершину.
32	G	IndexOutOfBoundsException	doAlgoNoStart().

	Start: U Finish: K		Проверка пути на корректную начальную вершину.

ЗАКЛЮЧЕНИЕ

В ходе Прохождения практики был изучен новый для участников команды язык программирования JAVA. Команда получила навыки работы на языке JAVA путём прохождения обучающего курса на платформе Stepik и созданию графического приложения с помощью библиотеки Swing и системы сборки Maven.

Было создано графическое Приложение, которое визуализирует работу алгоритма A* (A Стар).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хорстманн Core Java в 2х томах

2. Брюс Эккель Философия Java

3. Шилдт Java8 Полное руководство

4. Джошуа Блох Effective

5. Брайан Гоетц Java concurrency in practice

6. Стив Макконал Совершенный код, Роберт Мартин Чистый код и Чистая архитетура

7. Роберт Седжвик Алгоритмы на Java

8. https://docs.oracle.com/en/java/javase/16/

9. https://javarush.ru/groups/posts/isklyucheniya-java