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

ОТЧЕТ

по учебной практике

Тема: Алгоритм Борувки

Студент гр. 9303 _____ Лойконен М.Р.

Студент гр. 9303 _____ Микулик Д.П.

Студент гр. 9303 _____ Халилов Ш.А.

Руководитель _____ Ефремов М.А.

Санкт-Петербург

2021

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Лойконен М.Р., гр. 9303

Студент Микулик Д.П., гр. 9303

Студент Халилов Ш.А., гр. 9303

Тема работы: Алгоритм Борувки

Задание на практику:

Разработка визуализатора алгоритма Борувки на Java.

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Сроки проведения практики: 01.07.2021-14.07.2021

Дата сдачи отчета: 14.07.2021

Студент _____ Лойконен М.Р.

Студент _____ Микулик Д.П.

Студент _____ Халилов Ш.А.

Руководитель _____ Ефремов М. А.

АННОТАЦИЯ

Цель учебной практики заключается в разработке GUI-приложения, которое позволяет создать граф и визуализировать алгоритм Борувки для нахождения минимального остовного дерева в созданном графе.

Разработка программы происходит на языке Java командой из 3-х человек. Роли каждого из членов команды распределены в соответствии с имеющимися задачами. Сдача и демонстрация проекта происходит согласно плану разработки.

SUMMARY

The purpose of the training practice is to develop a GUI application that allows you to create a graph and visualize the Boruvka algorithm for finding the minimum spanning tree in the created graph.

The program is developed in Java by a team of 3 people. The roles of each of the team members are distributed in accordance with the existing tasks. Delivery and demonstration of the project takes place according to the development plan.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью практической работы является разработка графического приложения, выполняющего визуализацию работы алгоритма Борувки нахождения минимального остовного дерева графа.

При разработке приложения планируется реализация графического интерфейса для задания графа (напрямую, используя функционал приложения или из файла, где перечислены вершины и ребра графа). Также важной частью приложения является пошаговая визуализация алгоритма Борувки.

Разработка осуществляется на языке Java с использованием фреймворка Swing командой из трёх человек. В команде распределяются обязанности – разработка алгоритма, разработка интерфейса и визуализации, сборка и тестирование программы.

1. СПЕЦИФИКАЦИЯ ПРОГРАММЫ

1.1 ПРОТОТИП ИНТЕРФЕЙСА И USE-CASE ДИАГРАММА

Опишем изначальные требования к программе:

1. Программа представляет собой визуализацию алгоритма Борувки нахождения наименьшего остовного дерева.
2. Граф, на котором выполняется алгоритм, можно загружать из файла, а также создавать или модифицировать в самой программе.
3. Предполагается пошаговая визуализация алгоритма с возможностью отката на предыдущий шаг.
4. При визуализации алгоритма отображается дополнительная информация (например, разные компоненты связности окрашиваются в разные цвета).

USE-CASE диаграмма:

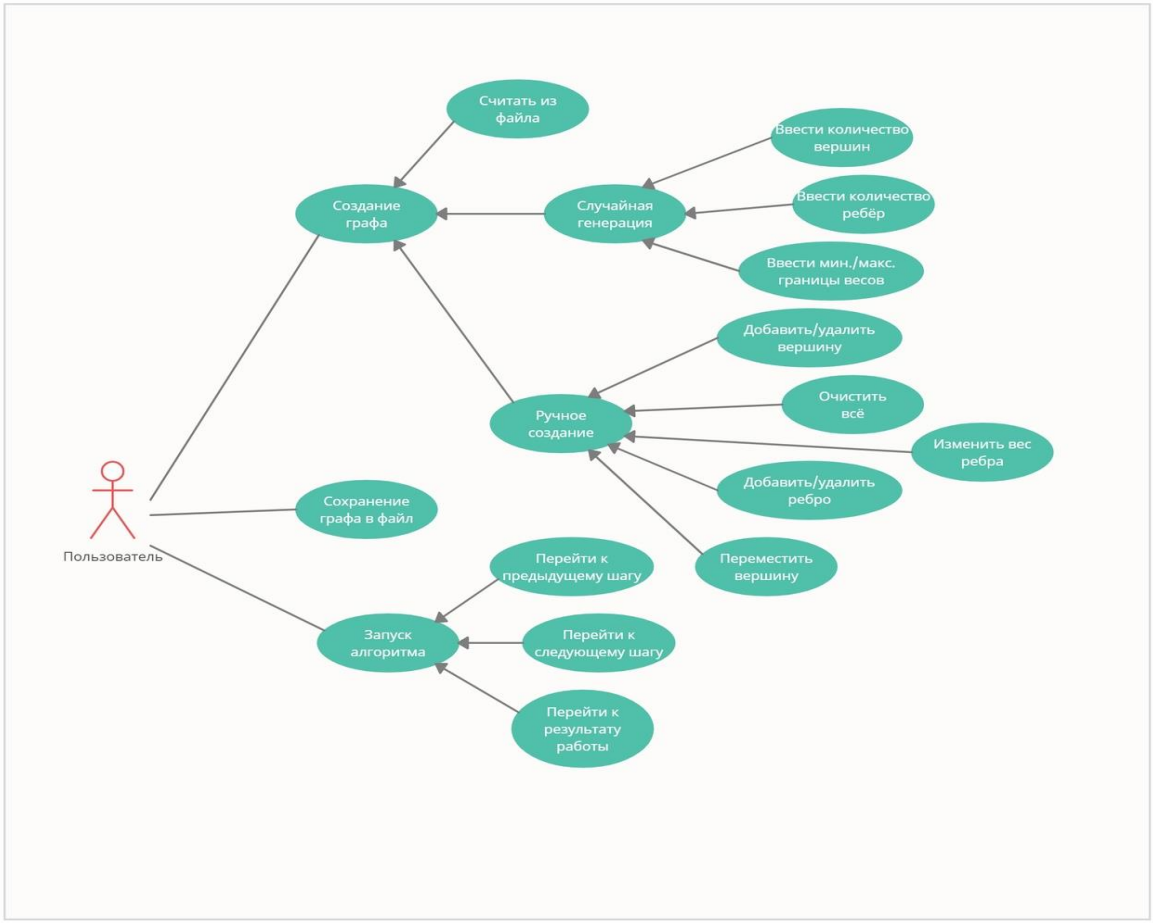

Рисунок 1 - USE-CASE диаграмма

Для реализации требований к функционалу, описанных в USE-CASE диаграмме, в прототип были внесены следующие элементы GUI:

- Три нижние кнопки на боковой панели инструментов позволяют реализовать функционал запуска алгоритма, получения результата работы алгоритма, а так же откат на предыдущий или следующие шаги работы алгоритма.
- Для возможности сохранения и загрузки графа в/из файла была добавлена панель меню, с соответствующим активным элементом «Файл».
- Генерация случайного графа также происходит при помощи вызова соответствующей команды из элемента «Файл» на панели меню.
- Для ручного создания графа служат три верхние кнопки боковой панели инструментов, которые позволяют добавить вершину или ребро, либо же очистить все.
- Созданы две сцены для визуализации графа и алгоритма.

Прототип интерфейса программы:

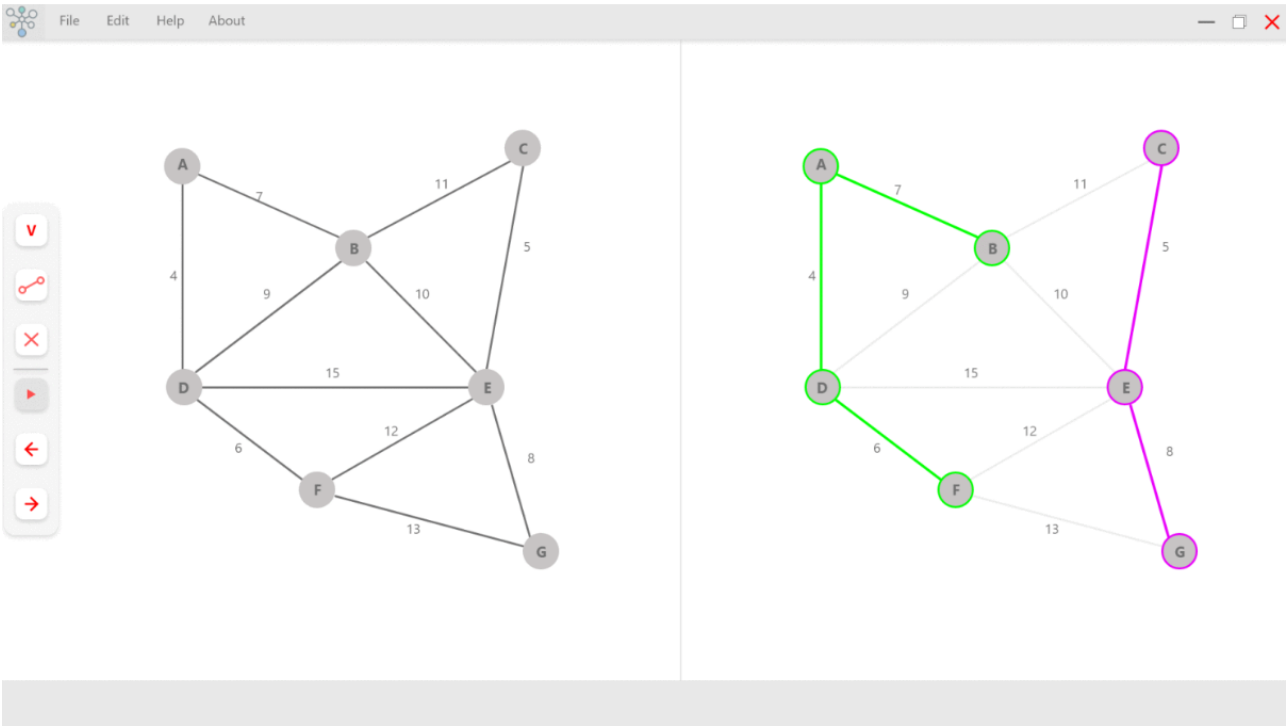

Рисунок 2- Прототип интерфейса программы

2. ОРГАНИЗАЦИЯ РАБОТЫ

2.1 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.

В соответствии с имеющимися задачами был составлен план разработки приложения:

1. К 02.07.2021 изучить основы Java для дальнейшей работы над проектом, создать репозиторий проекта.
2. К 03.07.2021 обсудить спецификацию программы (представить прототип окна GUI приложения, представление USE-CASE диаграммы).
3. К 06.07.2021 разработать интерфейс приложения, начать реализацию базового функционала приложения (реализация алгоритма Борувки), окончательно проработать архитектуру приложения, построить UML-диаграмму классов.
4. К 08.07.2021 реализовать базовый функционал приложения, связать реализацию функционала (базового, т.е. реализация и визуализация алгоритма Борувки) и интерфейса.
5. К 10.07.2021 расширить имеющийся функционал приложения (например, реализовать разные способы задания графа), убрать критичные ошибки и баги.
6. К 12.07.2021 произвести тестирование и отладку программы, при необходимости этапу тестирования может предшествовать рефакторинг имеющегося кода. Сдача проекта.

2.2 Распределение ролей в команде

Обязанности между участниками команды распределены следующим образом:

Лойконен М.Р. – реализация алгоритма Борувки, тестирование программы.

Микулик Д.П. – реализация классов, связующих интерфейс и бизнес-логику (в данном случае сам алгоритм Борувки), написание отчета.

Халилов Ш.А. – реализация графического интерфейса, сборка программы, реализация прототипа графического интерфейса.

3. ОПИСАНИЕ КОДА ПРОГРАММЫ

3.1 Описание алгоритма и используемых структур данных

Для понимания реализованных классов, реализующих логику приложения, требуется добавить некоторые теоретические сведения об алгоритме Борувки, а также описать подробнее классы, которые обеспечивают хранение графа.

Алгоритм Борувки - алгоритм поиска минимального остовного дерева во взвешенном неориентированном связном графе.

Алгоритм состоит из нескольких шагов:

1. Изначально каждая вершина графа G.— тривиальное дерево, а ребра не принадлежат никакому дереву.
2. Для каждого дерева T найдем минимальное инцидентное ему ребро. Добавим все такие ребра.
3. Повторяем шаг 2 пока в графе не останется только одно дерево T.

Для хранения графа были реализованы три класса: `Graph`, `Edge`, `Node`. Здесь `Graph` – основной класс, который представляет собой граф. Имеет следующие поля:

`ArrayList<Node> Vertices` – массив вершин графа.

`ArrayList<Edge> Edges` – массив ребер графа (каждое ребро имеет ссылки на стартовую и конечную вершины в массиве Vertices).

`Int CountVertices` – количество вершин графа.

`Int CountEdges` – количество ребер графа.

Реализованные методы данного класса представляют функционал для добавление и удаления вершин и ребер, а так же для получения состояния полей графа. Также был переопределен метод `clone()` для корректной создании копии графа.

Класс `Node` предназначен для хранения вершины графа. Имеет следующие поля:

`String name` – имя текущей вершины.

`Int component` – номер компоненты связности, к которой принадлежит текущая вершина.

`Int x` – координата x вершины на холсте.

`Int y` – координата y вершины на холсте.

Реализованные методы позволяют получать и задавать состояния имеющихся полей у класса вершины. Также переопределены методы `equals()`, `hashcode()` (для того, чтобы имелась возможность сравнения двух вершин) и `clone()`, для возможности создания копии вершины (для корректного сохранения состояние графа).

Класс `Edge` предназначен для хранения ребра графа. Имеет следующие поля:

`Node start` – хранит ссылку на стартовую вершину у ребра.

`Node end` – хранит ссылку на конечную вершину у ребра.

`Int weight` – хранит вес ребра.

Реализованные методы позволяют получать состояния полей класса ребра, а также изменять вес имеющегося ребра. Были переопределены методы

clone() для создания корректной копии ребра, методы equals() и hashcode() для корректного сравнения двух ребер а также метод toString() для вывода осмысленной информации о ребре.

3.2 Описание основных классов для реализации логики приложения

Для реализации функционала приложения были реализованы следующие классы:

- Graph, Edge, Node – классы, для хранения графа.
- Algorithm – общий интерфейс алгоритмов поиска МОД, BoruvkaAlg – класс, реализующий интерфейс Algorithm, МОД находится при помощи алгоритма Борувки. Основной метод, который позволяет сделать шаг алгоритма: algorithmStep().
- Facade – класс, обобщающий имеющуюся логику (граф, алгоритм, классы сохранения и команды).
- CareTaker, AlgorithmMemento – набор классов для реализации хранения промежуточных состояний при работе алгоритма (реализуют паттерн «Снимок»)
- Command, LoadCommand, GenerateCommand, VisualizeCommand – классы-команды, которые будут исполняться для загрузки, генерации и визуализации алгоритма Борувки.

Для реализации выбранной архитектуры приложения использовались следующие паттерны:

- Снимок – для реализации функционала сохранения шагов алгоритма. Реализован с помощью классов: AlgorithmMemento (является вложенным классом для BoruvkaAlg), который сохраняет состояние алгоритма, CareTaker, который предоставляет методы backup() и

`undo()` для сохранения промежуточного состояния алгоритма и отката к предыдущему состоянию.

- Команда – для реализации основных команд, не связанных с ручным созданием графа, а именно: команда по загрузке графа из файла (класс `LoadCommand`), генерации графа (класс `GenerateCommand`), запуску визуализации алгоритма (класс `VisualizeCommand`).
- Фасад – для обобщения всех классов, которые реализуют логику (классов графа, алгоритма, команды и сохранения состояния).

Ниже для конкретного понимания реализованных классов и их взаимосвязи представлена UML диаграмма классов.

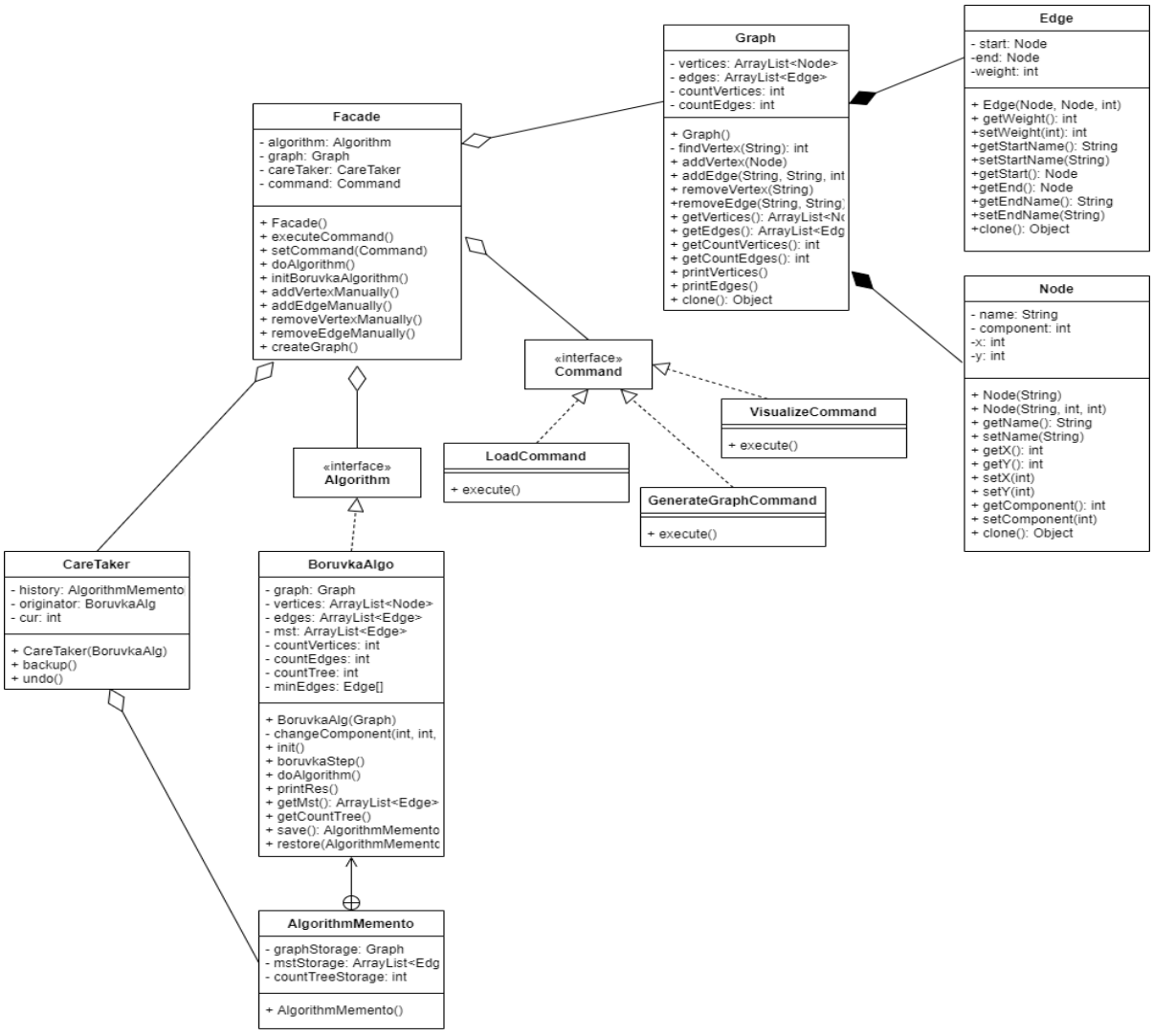

Рисунок 3 - UML диаграмма классов

4. ТЕСТИРОВАНИЕ ПРОГРАММЫ

Для проверки правильности реализованной логики работы алгоритма был реализован набор тестов, который представлен в таблице ниже. Здесь под графом G подразумевается следующий граф: граф-треугольник, с тремя вершинами (a, b, c), ребрами (a, b) с весом 3, (a, c) с весом 1, (b, c) с весом 4.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1	Граф G	true	Проверка метода поиска вершины b в графе (findExistingVertex())
2	Граф G	true	Проверка того, что в графе нет вершины g (findExistingVertex())
3	Граф G	IllegalArgumentException	Проверка генерации исключения при проверке наличия null вершины в графе.
4	Граф G	true	Проверка добавления вершины k в граф (addVertex())
5	Граф G	IllegalArgumentException	Проверка добавления null в качестве вершины графа
6	Граф G	true	Проверка добавления нового ребра (addEdge())
7	Граф G	Без изменений	Проверка добавления уже существующего ребра (addEdge())
8	Граф G	IllegalArgumentException	Добавление ребра с вершинами null
9	Граф G	IllegalArgumentException	Добавление ребра с нулевым весом
10	Граф G	true	Удаление существующей вершины (removeVertex())
11	Граф G	IllegalArgumentException	Удаление несуществующей вершины (removeVertex())
12	Граф G	IllegalArgumentException	Удаление null- вершины (removeVertex())
13	Граф G	true	Удаление существующего ребра (removeEdge())

14	Граф G	IllegalArgumentException	Удаление несуществующего ребра (removeEdge())
15	Граф G	IllegalArgumentException	Удаление null-ребра (removeEdge())
16	Граф G	true	Удаление ребра-петли (removeEdge())
17	Граф G	true	Получение списка вершин (GetVertices())
18	Граф G	true	Получение списка ребер (GetEdges())
19	Граф G	true	Получение числа вершин (GetCountVertices())
20	Граф G	True (число вершин уменьшилось на 1)	Получение числа вершин после удаления вершины (GetCountVertices())
21	Граф G	True (число вершин не изменилось)	Получение числа вершин после удаления ребра (GetCountVertices())
22	Граф G	true	Получение числа ребер (GetCountEdges())
23	Граф G	True (число ребер не иззменилось)	Получение числа ребер после удаления вершины (GetCountEdges())
24	Граф G	True (число ребер уменьшилось на 1)	Получение числа ребер после удаления ребра (GetCountEdges())
25	Граф G	True (получено корректное МОД)	Проверка работы алгоритма (doAlrorithm())

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ