

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. 8303	_____	Абибулаев Э.Э.
Студент гр. 8303	_____	Рудько Д.Ю.
Студент гр. 8303	_____	Парфентьев Л.М.
Руководитель	_____	Ефремов М.А.

Санкт-Петербург
2020

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Абибулаев Э.Э. группы 8303

Студент Рудько Д.Ю. группы 8303

Студент Парфентьев Л.М. группы 8303

Тема практики: Алгоритм Борувки

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: Борувки.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 00.07.2020

Дата защиты отчета: 00.07.2020

Студент гр. 8303

Абибулаев Э.Э.

Студент гр. 8303

Рудько Д.Ю.

Студент гр. 8303

Парфентьев Л.М.

Руководитель

Ефремов М.А.

АННОТАЦИЯ

Целью данной учебной практики является разработка графического приложения визуализации алгоритма Борувки поиска минимального остовного дерева в графе. Приложение пишется на языке Java с использованием фреймворка JavaFX.

Приложение разрабатывается бригадой из трех человек за несколько итераций.

SUMMARY

The purpose of this educational practice is to develop a graphical application for visualizing the Boruvka algorithm for finding the minimum spanning tree in a graph. The application is written in Java using the JavaFX framework.

The application is developed by a team of three people for several iterations.

ОГЛАВЛЕНИЕ

1. Требования к программе.....	5
1.1. Начальные требования.....	5
Диаграмма сценариев использования:.....	6
2. План разработки и распределение ролей в команде.....	6
2.1. План разработки.....	6
2.2. Распределение ролей в бригаде.....	7
3. Особенности реализации.....	8
3.1. Структура проекта.....	8
Диаграммы классов.....	9

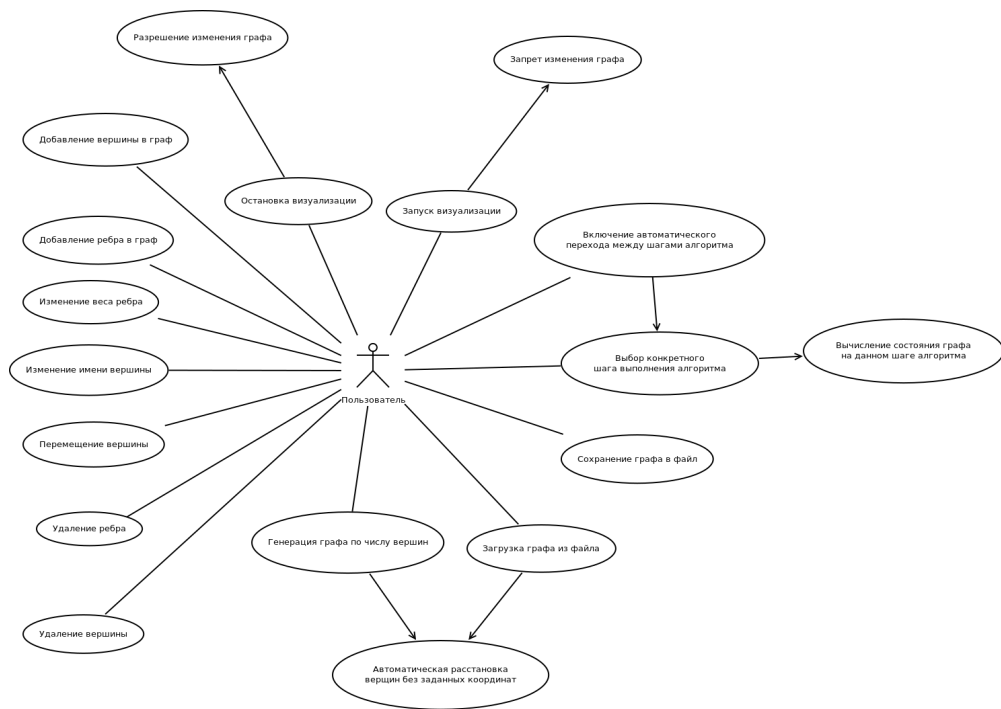
1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Начальные требования

- Программа представляет собой визуализацию алгоритма Борувки нахождения наименьшего остовного дерева.
- Граф, на котором выполняется алгоритм, можно загружать из файла, а также создавать или модифицировать в самой программе.
- Визуализация пошаговая: на каждом шаге может происходить следующее:
 - Выбор очередного ребра;
 - Объединение компонент связности.
- Граф можно загрузить из файла и сохранить в файл.
 - При загрузке у части вершин могут быть указаны координаты. У остальных вершин координаты должны выставляться автоматически.
 - При сохранении графа координаты вершин записываются в файл.
- Перед запуском алгоритма граф можно изменять.
 - У вершин можно менять текст.
 - У ребер можно менять веса.
 - Можно добавлять новые ребра и вершины.
 - Можно удалять ребра и вершины.
- После запуска алгоритма граф становится неизменяемым. При выходе из режима визуализации граф снова становится изменяемым.

- При визуализации алгоритма отображается дополнительная информация.
 - Вершины и рёбра из разных компонент связности раскрашены разными цветами. Цвета выбираются случайным образом при запуске алгоритма.
 - При отображении каждого шага, на котором выбирается новое ребро, это ребро подсвечивается.
 - Не выбранные ребра внутри компонент связности (т.е. такие ребра, которые точно не будут выбраны на последующих шагах), будут отображаться специальным образом (тонкими светлыми линиями).

Диаграмма сценариев использования:



2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В КОМАНДЕ

2.1. План разработки

- 2 июля: распределение ролей в бригаде; UML-диаграмма сценариев использования.

- 4 июля: графический интерфейс (не рабочий), проектирование классов программы, проектирование поведения программы.
- 6 июля: случайная генерация входных данных, обычная реализация алгоритма (до конца без промежуточных результатов) с отображением результата, план тестирования.
- 8 июля: прототип визуализации, тестирование, пошаговая реализация алгоритма.
- 10 июля: подготовка итогового “релиза”, завершение отчета.

2.2. Распределение ролей в бригаде

- Абибулаев Э.Э.: разработка визуализации и графического интерфейса.
- Рудько Д.Ю.: реализация алгоритма.
- Парфентьев Л.М.: тестирование и сборка приложения.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структура проекта

- Класс, реализующий граф – `Graph`. Вспомогательные классы `Graph.Edge` и `Graph.Node` представляют рёбра и узлы графа.
- Класс, содержащий реализацию алгоритма – `BoruvkaAlgorithm`. Этот класс получает граф в конструкторе.
- Промежуточные состояния, которые отображаются при визуализации, являются объектами `BoruvkaSnapshot`. Эти объекты содержат всю необходимую информацию, такую как тип шага и состав компонент связности.
- Промежуточные состояния могут соответствовать шагам различного типа. Шаги представлены абстрактным классом `BoruvkaStep`, а также его наследниками.
- Из объекта алгоритма промежуточные состояния запрашиваются как из итератора – методом `next`. Метод `hasNext` проверяет, что остались ещё не отданные состояния.
- Алгоритм может вычислять все состояния сразу (при вызове конструктора или первом вызове `next`), а может вычислять их по мере необходимости,
- Компоненты связности, образующиеся при выполнении алгоритма, представлены классом `Group`.
- Класс отображения – `ConcreteGraphView`. Он получает обновления графа через интерфейс `Subscriber`. Отображение отвечает за отрисовку графа и шагов алгоритма на экране. Промежуточные

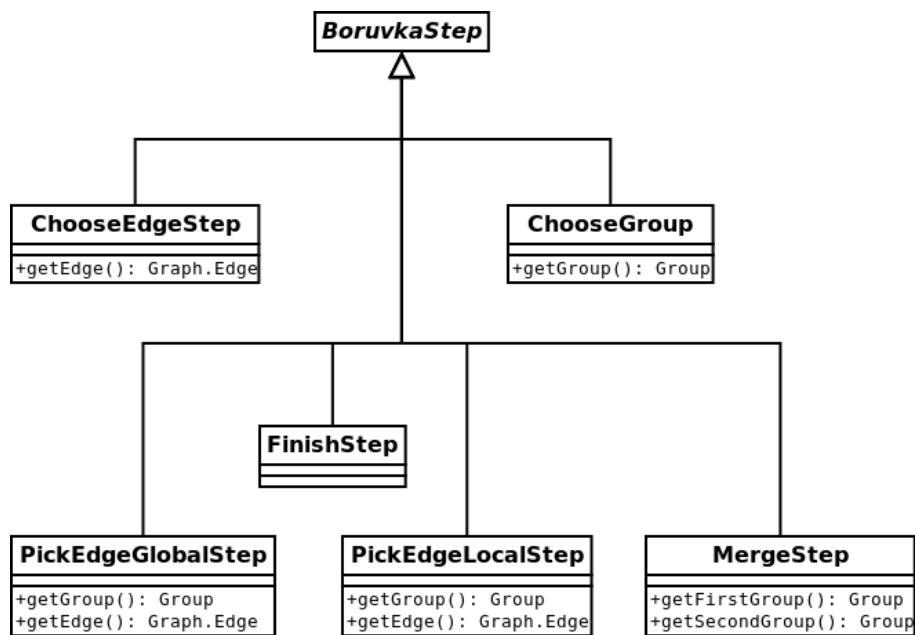


Рисунок 2: Диаграмма классов шагов алгоритма

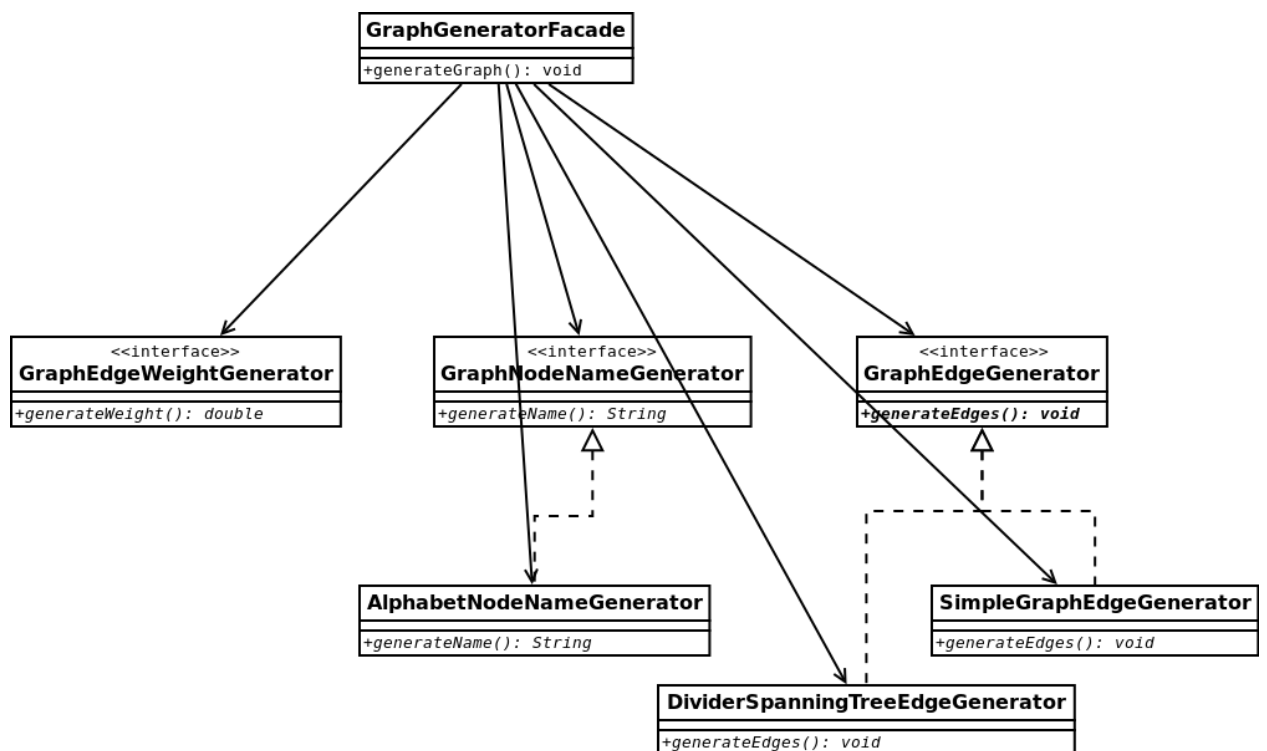


Рисунок 3: Диаграмма классов генерации графов

3.2 Описание алгоритма

Алгоритм Борувки работает следующим образом:

1. Изначально, граф, который в итоге станет минимальным каркасом исходного графа, содержит все его вершины, и не содержит ни одного ребра. Таким образом, каждая вершина составляет отдельную компоненту связности.
2. На каждом шаге, из исходного графа для каждой компоненты связности выбирается самое дешёвое ребро, ведущее в другую компоненту. Затем, эти рёбра добавляются в каркас.
3. Алгоритм завершает работу когда ни для одной компоненты связности не нашлось подходящего ребра.

3.3 Диаграммы состояний

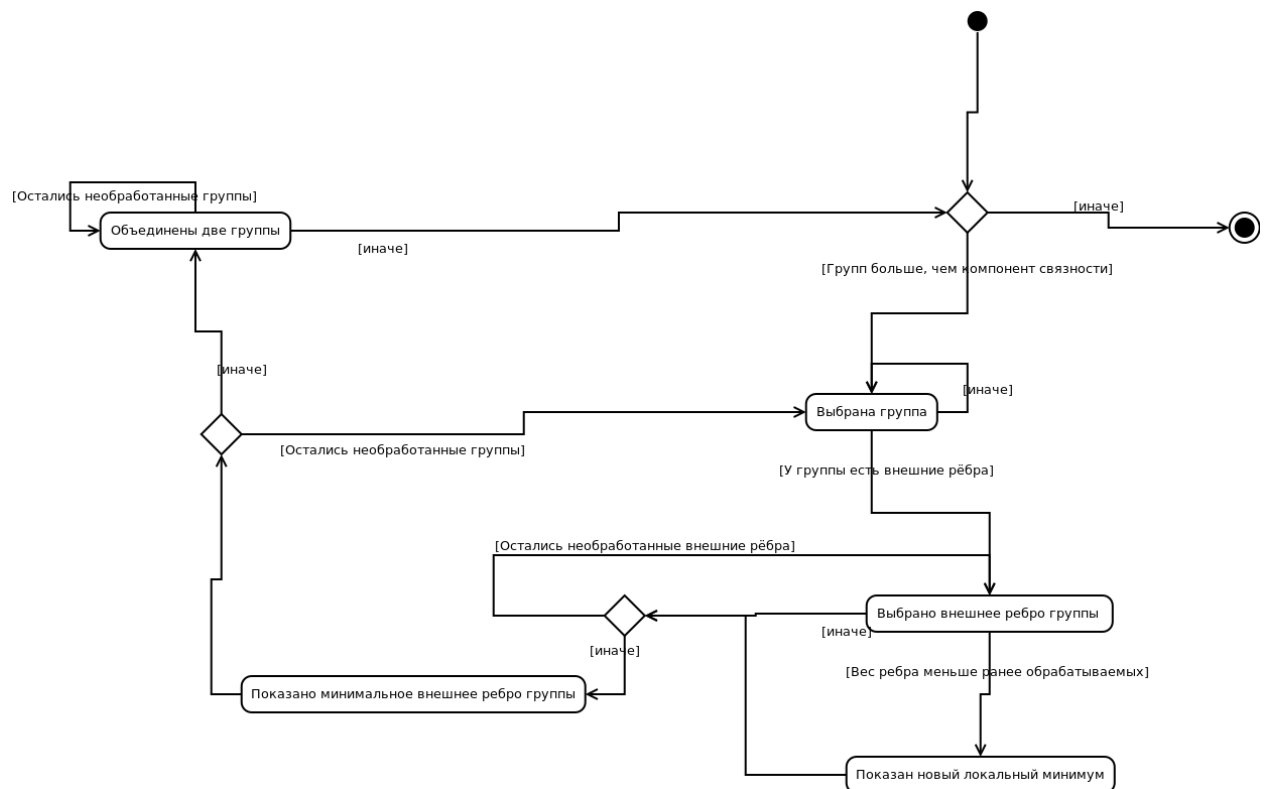


Рисунок 4: Диаграмма состояний визуализации

На рисунке 4 показана диаграмма состояний визуализации алгоритма. Состояния соответствуют подклассам `BoruvkaStep` с рисунка 2.

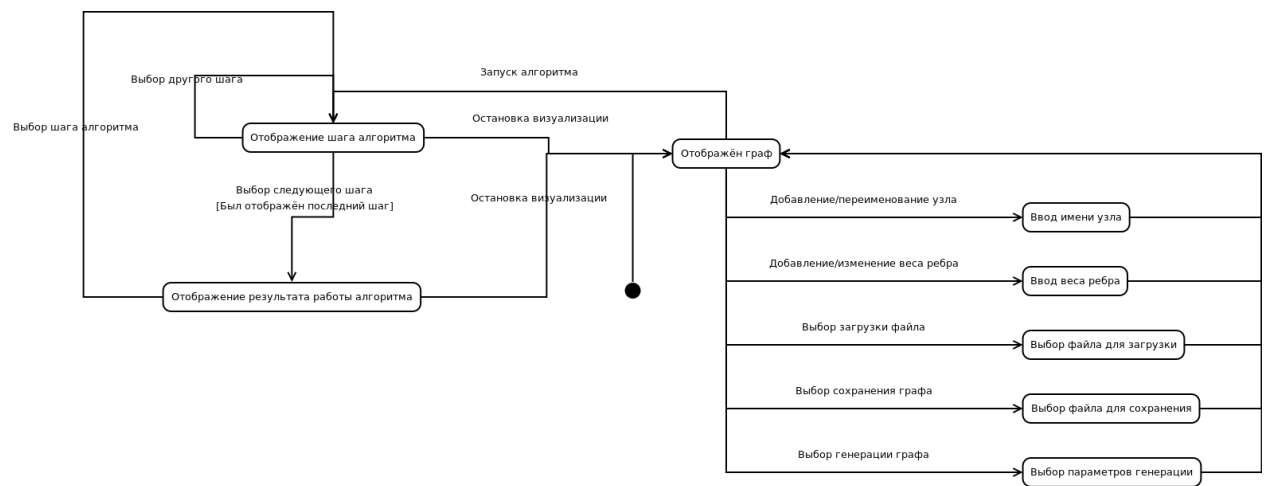


Рисунок 5: Диаграмма состояний пользовательского интерфейса

4. ТЕСТИРОВАНИЕ

4.1. План тестирования

- Тестирование операций графа:
 - Базовый функционал
 - Добавление ребра в несуществующую вершину, запрос рёбер несуществующей вершины и т. п. должны выдавать исключение.
 - Удаление вершины должно вызывать удаление связанных с ней рёбер.
 - При перечислении рёбер через `getEdges` не должно быть дубликатов
- Тестирование алгоритма:
 - Базовый функционал
 - Корректная обработка неоднозначных ситуаций (два ребра с одинаковым весом: поведение будет явно задано в спецификации)
 - Корректная обработка несвязных графов (требует верного определения условия завершения)