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

ОТЧЕТ

по учебной практике

Тема: Визуализация алгоритмов на языке Kotlin

Студент гр. 8381 _____ Сосновский Д.Н.

Студент гр. 8381 _____ Киреев К.А.

Студент гр. 8381 _____ Муковский Д.В.

Руководитель _____ Фирсов М.А.

Санкт-Петербург
2020

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Сосновский Д.Н. группы 8381

Студент Киреев К.А. группы 8381

Студент Муковский Д.В. группы 8381

Тема практики: визуализация алгоритмов на языке Kotlin

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Kotlin с графическим интерфейсом.

Алгоритм: Дейкстры.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 10.07.2020

Дата защиты отчета: 10.07.2020

Студент	_____	Сосновский Д.Н.
Студент	_____	Киреев К.А.
Студент	_____	Муковский Д.В.
Руководитель	_____	Фирсов М.А.

АННОТАЦИЯ

В ходе выполнения задания учебной практики была реализована программа, предназначенная для нахождения кратчайшего пути в графе. Поиск кратчайшего пути осуществляется с использованием алгоритма Дейкстры. Разработанная программа детально показывает этапы работы алгоритма при построении кратчайшего пути. Программа была написана на языке программирования Kotlin, в среде разработки IntelliJ Idea.

SUMMARY

In the course of the assignment, a program was developed designed to find the shortest paths in a graph. Search using the Dijkstra algorithm. The developed program shows in detail the stages of the algorithm when building the shortest path. Kotlin, in the IntelliJ Idea development environment.

СОДЕРЖАНИЕ

1.	Постановка задачи	6
1.1.	Формулировка задания	6
1.2.	Формальное определение и сложность алгоритма	6
1.3.	Реализуемый алгоритм	6
1.4	Псевдокод	7
2.	Спецификация	8
2.1.	Требования к интерфейсу пользователя	8
2.2.	Требования к вводу исходных данных	10
2.3.	Требования к визуализации	10
2.4.	Уточнение требований после сдачи первой версии	11
3.	План разработки и распределение ролей в бригаде	12
3.1.	План разработки	12
3.2.	Распределение ролей в бригаде	13
4.	Особенности реализации	14
4.1	Структуры данных	14
4.2	Основные методы	14
5.	План тестирования	15
5.1.	Тестирование графического интерфейса	15
5.2.	Тестирование алгоритма	16
5.3.	Unit тестирование	19
	Заключение	20
	Список использованных источников	21
	Приложение А. Исходный код	22

ВВЕДЕНИЕ

Разработка программы будет вестись с использованием языка программирования Kotlin и его возможностей по созданию GUI. Для написания программы используется интегрированная среда разработки IntelliJ IDEA.

Цели выполнения учебной практики:

1. Освоение среды программирования Kotlin, особенностей и синтаксиса данного языка.

2. Изучение средств для реализации графического интерфейса, а именно – библиотеки Swing.

3. Получение таких навыков как:

- разработка программ на языке Kotlin;
- работа с системой контроля версий, репозиториями;
- командная работа.

Поставленные цели будут достигнуты во время выполнения проекта (визуализации алгоритма Дейкстры), выбранного бригадой.

Алгоритм Дейкстры - алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году. Данный алгоритм находит кратчайшие пути от одной из вершин графа до всех остальных вершин. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании и технологиях, например, при планировании автомобильных и авиамаршрутов, при разводке электронных плат, в протоколах маршрутизации.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Формулировка задания

Разработать программу, которая визуализирует алгоритм Дейкстры для поиска кратчайших путей в графе на языке программирования Kotlin.

1.2. Формальное определение и сложность алгоритма

Дан взвешенный ориентированный граф G(V,E) без дуг отрицательного веса. Найти кратчайшие пути от некоторой вершины a графа G до всех остальных вершин этого графа.

Сложность алгоритма Дейкстры зависит от способа нахождения вершины v, а также способа хранения множества непосещённых вершин и способа обновления меток. В простейшем случае, когда для поиска вершины с минимальным d[v] просматривается всё множество вершин, а для хранения величин d используется массив, время работы алгоритма есть O(n^2).

1.3. Реализуемый алгоритм

Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Инициализация.

Метка самой вершины a полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от a до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.

Шаг алгоритма.

Если все вершины посещены, алгоритм завершается.

В противном случае, из ещё не посещённых вершин выбирается вершина u, имеющая минимальную метку.

Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из u, назовём

соседями этой вершины. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещённую и повторим шаг алгоритма.

1.4. Псевдокод

```
func dijkstra(s):
    for v ∈ V
        d[v] = ∞
        used[v] = false
    d[s] = 0
    for i ∈ V
        v = null
        for j ∈ V                          // найдём вершину с минимальным расстоянием
            if !used[j] and (v == null or d[j] < d[v])
                v = j
        if d[v] == ∞
            break
        used[v] = true
        for e : исходящие из v рёбра      // произведём релаксацию по всем рёбрам, исходящим из v
            if d[v] + e.len < d[e.to]
                d[e.to] = d[v] + e.len
```

2. СПЕЦИФИКАЦИЯ

2.1. Требования к интерфейсу пользователя

Пользовательский интерфейс должен представлять собой диалоговое окно, содержащее набор кнопок, предназначенных для управления состоянием программы.

Диалоговое окно должно состоять из:

- Рабочей области для построения графа.
- Кнопки «Сохранить», которая должна позволять пользователю сохранить созданный в рабочей области граф в виде .txt файла.
- Кнопки «Загрузить», которая должна позволять пользователю загрузить граф, для которого необходимо применить алгоритм Дейкстры, в виде .txt файла.
- Кнопки «Сгенерировать граф», которая должна генерировать случайный граф в рабочей области.
- Кнопки «Вперед», которая должна отобразить следующую итерацию алгоритма.
- Кнопки «Назад», которая должна отобразить предыдущую итерацию алгоритма

Первоначальный макет представлен на рис. 1

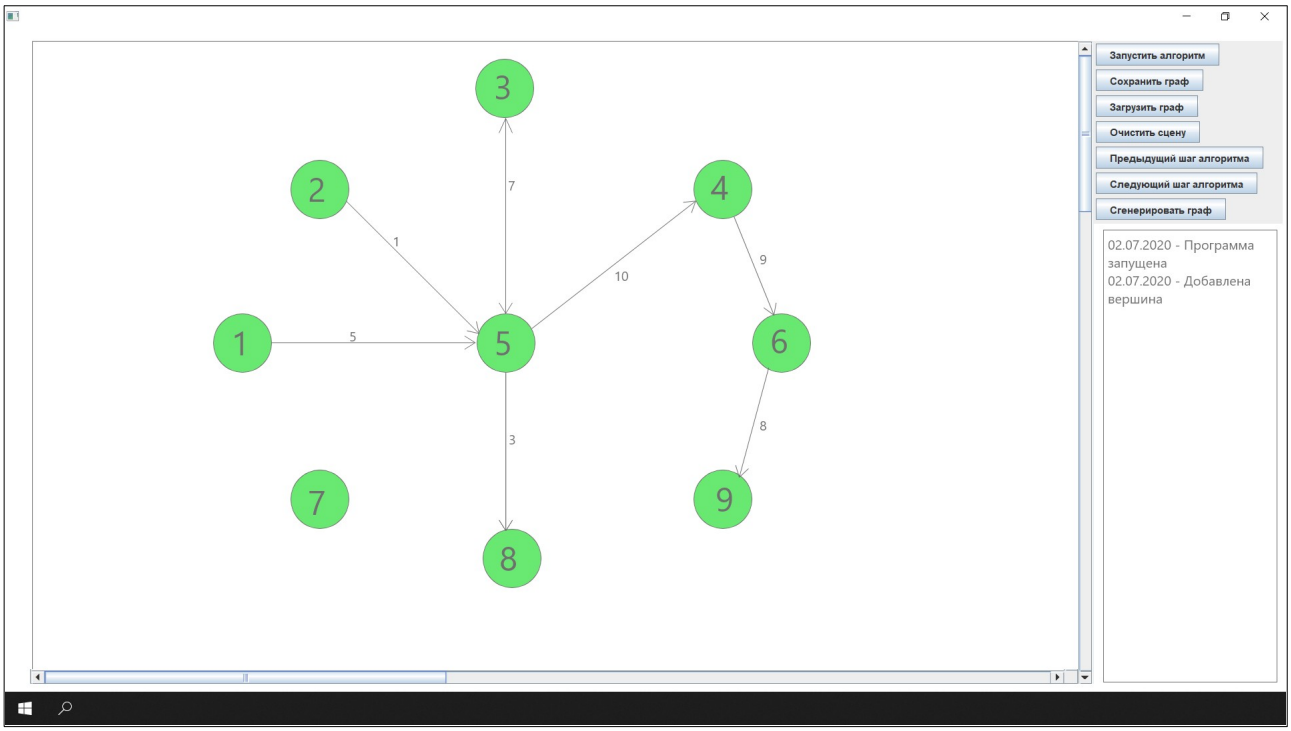

Рисунок 1 — Макет программы

Также представлена use case диаграмма на рисунке 2.

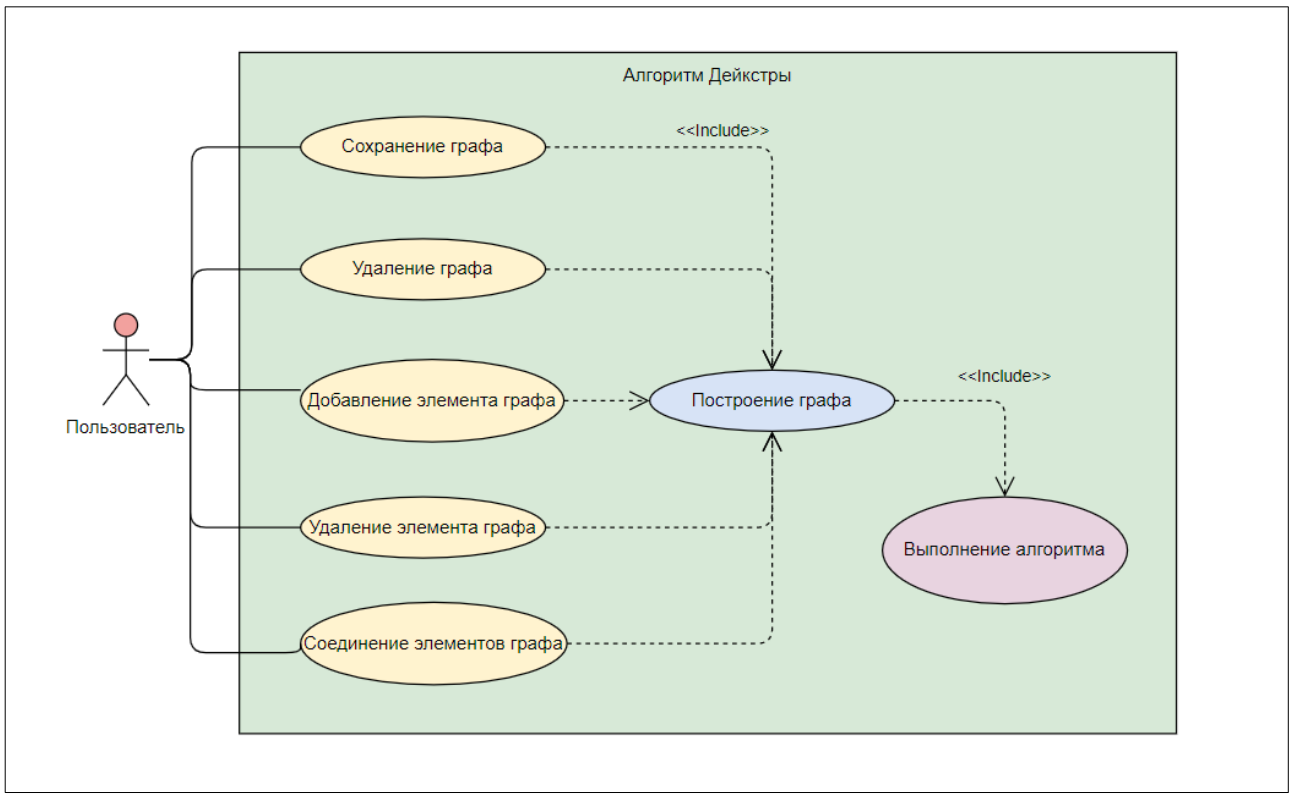

Рисунок 2 - Use case диаграмма

2.2. Требования к вводу исходных данных

На вход алгоритму должен подаваться взвешенный ориентированный граф. Именем вершины могут быть числа. Область создания графа предоставляет возможность ввести данные вручную, с файла или сгенерировать. При ручном вводе происходит взаимодействие с графическим представлением графа с помощью мыши. Двойным кликом по области создается вершина с указанным именем, а через контекстное меню вершины можно будет удалить ее или провести ребро к другой вершине. При считывании с файла надо нажать соответствующую кнопку. Формат данных для считывания из файла: на первой строке находятся координаты вершин а их именем является порядок, в котором они расположены в файле и далее на второй строке список попарных вершин с их весами, обозначающих ребра. При генерации нужно ввести необходимые данные в поля генерации.

2.3. Требования к визуализации

Перед началом работы алгоритма создается граф, одним из указанных выше способов. Также уже существующие вершины можно перемещать. При выделении вершины ее цвет меняется на специальный цвет. На каждом шаге будут выделяться все вершины, с которыми сейчас происходит действие алгоритма в специальный отличный цвет, будет выводится дополнительный информационный текст, о том что на этом шаге происходит, с какими вершинами и ребрами идет взаимодействие, а также будут подсвечиваться используемые ребра. В конце появляется всплывающее окно с результатами работы алгоритма.

2.4. Уточнение требований после сдачи первой версии

- Конечную вершину указывать не надо: алгоритм Дейкстры находит расстояния до всех вершин от стартовой.
- После завершения алгоритма должна быть возможность отредактировать граф и запустить алгоритм заново (возможно, с другой стартовой вершиной).

3. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

3.1. План разработки

Репозиторий с исходным кодом и данной спецификацией расположен на GitHub по адресу: https://github.com/DmitriySosnovskiy/kotlin_training_practice

План разработки программы разбит на следующие шаги:

1. Создание прототипа (03.07): к этому шагу будет создано приложение, демонстрирующее интерфейс, но почти не реализующее основные функции;

2. Первая версия (06.07): на этом шаге будет реализован класс, отвечающий за работу самого алгоритма Дейкстры. Затем он будет соединен с классом GUI, используя интерфейсы обоих классов; в результате получится начальная рабочая версия программы; реализовано взаимодействие с вершинами для ручного ввода; написание тестов к проекту;

Перед релизом первой версии планируется тестирование основных функций работы с GUI (корректная работа основных кнопок, которые планируются к данной версии (запустить алгоритм, очистить сцену)), тестирование алгоритма Дейкстры на данных с ручного ввода (некоторые примеры описаны в плане тестирования).

3. Вторая версия (08.07): реализация пошаговой работы программы. реализация выделения вершин при исполнении шагов алгоритма; реализован ввод из файла; реализован вывод логов;

Перед релизом второй версии планируется тестирование дополнительных функций работы с GUI (корректная работа кнопок сохранить, загрузить, шаги), тестирование корректного вывода логов.

4. Третья (конечная) версия (10.07): добавление возможности генерации входных данных. Реализация вывода дополнительного информационного текста при шагах алгоритма, а также выделение ребер. Исправление ошибок проекта. Тестирование проекта.

Перед релизом финальной версии планируется тестирование корректной генерации входных данных, тестирование окна с информацией, выводимого на шагах алгоритма, тестирование конечного варианта GUI, финальное тестирование алгоритма Дейкстры, используя JUnit 5.

3.2. Распределение ролей в бригаде

Роли в бригаде распределены следующим образом:

- Сосновский Дмитрий: реализация GUI и методов взаимодействия с ним;
- Киреев Константин: реализации алгоритма Дейкстры и методов взаимодействия с ним; тестирование приложения; написание отчета.
- Муковский Даниил: реализация архитектуры программы с использованием паттерна MVP (model view presenter); реализация части presenter и model; проектирование системы классов и их взаимодействия;

4. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

4.1. Структуры данных

4.2. Основные методы

5. ПЛАН ТЕСТИРОВАНИЯ

5.1. Тестирование графического интерфейса

- Для тестирования используется jar файл.

1. Добавление n вершин, двойным нажатием на холсте. На экране по порядку нажатия появляются вершины с номерами от 1 до n. Вершины можно передвигать для удобства восприятия.

<Рисунок тестирования 1>

2. Добавление ребер. При нажатии на вершину в контекстном меню задается вес ребра. Далее проводится ребро до второй вершины. На холсте появляются ребра с их весами в центре.

<Рисунок промежуточного результата тестирования 2>

<Рисунок конечного результата тестирования 2>

3. Нажатие на кнопку «Запустить алгоритм», которая запускает алгоритм Дейкстры. Исходная вершина должна менять цвет (в финальной версии).

<Рисунок тестирования 3>

4. Нажатие на кнопку «Следующий шаг алгоритма» на графической сцене. Подсвечивается текущая вершина работы алгоритма (в финальной версии). Нажатие происходит пока алгоритм не закончит свою работу.

<Рисунок промежуточного результата тестирования 4>

<Рисунок промежуточного результата тестирования 4>

<Рисунок промежуточного результата тестирования 4>

Алгоритм закончил работу, вывод соответствующего сообщения в всплывающем окне.

<Рисунок конечного результата тестирования 4>

5. Нажатие на кнопку «Очистить сцену». Текущий граф исчезает и сцена становится пустой.

<Рисунок тестирования 5>

6. Нажатие на кнопку «Загрузить граф». Далее из выбранного файла загружается граф и выводится на сцену.

<Рисунок тестирования 6>

7. Нажатие на кнопку «Сохранить граф». Далее в выбранный файл сохраняется граф.

<Рисунок тестирования 7>

5.2. Тестирование алгоритма

План тестирования алгоритма Дейкстры представлен в таблице 1.

Таблица 1 - Тестирование алгоритма

№	Начальные данные	Результат
1	1 2 7 1 3 9 1 6 14 2 3 10 2 4 15 3 4 11 3 6 2 4 5 6 5 6 9	Кратчайший путь из 1 в 6: 11 1 → 3(9) → 6(11)
2	Удалим ребра [1 3] и [2 4], удалим вершину и запустим алгоритм	Кратчайший путь из 1 в 6: 14 1 → 6
3	После запуска алгоритма попробуем нажать на кнопку «Запуск алгоритма»	Ошибка: кнопка недоступна; алгоритм уже запущен
4	После завершения алгоритма попробуем нажать на кнопку «Следующий шаг алгоритма»	Ошибка: кнопка недоступна; алгоритм уже закончил работу
5	Добавим вершину, не связывая её с графом, и запустим алгоритм	Ошибка: некорректный граф; граф должен быть связным

Продолжение таблицы 1

6	Начальные данные: 1 2 7 1 3 9 1 6 14 2 3 10 2 4 15 3 4 11 3 6 2 4 5 6 5 6 9 1 1 1	Ошибка: некорректный граф; нельзя создавать петли
7	Начальные данные: 1 2 1 1 3 4 2 4 2 3 1 1 1 3.2 5	Ошибка: некорректный граф; концы и вес ребра должны задавать натуральными числами
8	1 2 7 1 6 14 2 3 10 2 8 6 8 10 9 10 9 2 9 11 6 11 12 4 12 13 5 5 13 18 5 6 9 2 6 2 10 3 1 3 4 11 9 4 1 4 5 6 4 13 9 4 12 8 4 11 4	Кратчайший путь из 1 в 13: 34 1 → 2 → 8 → 10 → 9 → 4 → 13

Продолжение таблицы 1

9	1 2 7 1 6 14 2 3 10 2 8 6 8 10 9 10 9 2 9 11 6 11 12 4 12 13 5 5 13 18 5 6 9 2 6 2 10 3 1 3 4 11 9 4 1 4 5 6 4 13 9 4 12 8 4 11 4 10 14 1 14 15 1 15 16 1 16 17 1 17 13 1	Кратчайший путь из 1 в 13: 27 1 → 2 → 8 → 10 → 14 → 15 → 16 → 17 → 13
10	Попробуем добавить ребро к вершинам уже имеющим ребро	Ошибка: вершины уже имеют ребро
11	Запустим алгоритм, не создавая графа	Ошибка: для запуска алгоритм нужен граф, состоящий минимум из двух вершин
12	Не запуская алгоритма, нажмем на кнопку «Следующий шаг алгоритма»	Ошибка: чтобы сделать шаг, запустите алгоритм
13	Нажмем на кнопку «Сохранить граф» без созданного графа	Ошибка: чтобы сохранить граф, сперва создайте граф
14	Начальные данные: 1 2 -1	Ошибка: вес ребра должен задаваться натуральным числом

5.3. Unit тестирование

При написании unit тестирования будут созданы классы, в которых выполняется проверка правильности реализации методов. При запуске тестов сначала создается экземпляр тест-класса (для каждого теста в классе отдельный экземпляр класса), затем выполняется метод, который запускает сам тест. Если какой-либо из методов выбрасывает исключение, тест считается провалившимся.

При сравнении ожидаемых и реальных результатов работы методов будет использоваться платформа JUnit 5.

ЗАКЛЮЧЕНИЕ

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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ПРИЛОЖЕНИЯ

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