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

ОТЧЕТ

по учебной практике

Тема: Алгоритм ЯПД (Прима)

Студент гр. 9304 _____ Попов Д.С.

Студентка гр. 9304 _____ Рослова Л.С.

Студентка гр. 9304 _____ Паутова Ю.В.

Руководитель _____ Жангиров Т.Р.

Санкт-Петербург

2021

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Попов Д.С. группы 9304

Студентка Рослова Л.С. группы 9304

Студентка Паутова Ю.В. группы 9304

Тема практики: Алгоритм ЯПД (Прима)

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Алгоритм: ЯПД (Прима).

Сроки прохождения практики: 01.07.2021 – 14.07.2021

Дата сдачи отчета: 08.07.2021

Дата защиты отчета: 09.07.2021

Студент		Попов Д.С.
Студентка		Рослова Л.С.
Студентка		Паутова Ю.В.
Руководитель		Жангиров Т.Р.

АННОТАЦИЯ

В период прохождения данной учебной практики реализуется командная итеративная разработка визуализатора алгоритма ЯПД на языке программирования Java с графическим интерфейсом. Алгоритм ЯПД (Прима) – алгоритм построения минимального остового дерева взвешенного связанного неориентированного графа. Разработанное в ходе работы приложение визуализирует пошаговое выполнение данного алгоритма.

SUMMARY

During the course of this training practice, a team iterative development of a visualizer of the YAPD algorithm in the Java programming language with a graphical interface is implemented. The YAPD algorithm (Prima) is an algorithm for constructing a minimal spanning tree of a weighted connected undirected graph. The application developed during the work visualizes the step-by-step execution of this algorithm.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной учебной практики является изучение алгоритма ЯПД и итеративная разработка его визуализатора на языке программирования Java с графическим интерфейсом.

Алгоритм ЯПД (Прима) выглядит следующим образом:

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Реализация алгоритма	Сложность алгоритма
Тривиальная	$O¿$
Сортировка инцидентных рёбер каждой вершины по возрастанию весов	$O¿$
Хранение для каждой невыбранной вершины минимального по весу ребра, соединяющего её с уже выбранной	$O(\|V\|^2)$

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требования к реализации алгоритма

- Алгоритм должен быть реализован так, чтобы можно было использовать любой тип данных.
- Алгоритм должен поддерживать возможность включения промежуточных выводов и пошагового выполнения.

1.1.2. Требования к проекту

- Возможность запуска через GUI и по желанию CLI (достаточно промежуточных выводов).
- Загрузка данных из файла или ввод через интерфейс.
- GUI должен содержать интерфейс управления работой алгоритма, визуализацию алгоритма, окно с логами работы.
- Возможность запуска алгоритма заново на новых данных без перезапуска программы.
- Возможность продвижения/отката на один шаг, завершения алгоритма до конца.
- Возможность сброса алгоритма в исходное состояние.

1.2. Уточнение требований после сдачи прототипа

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Итерация	Дата сдачи	Требования
Сдача прототипа	06.07.2021	• Эскиз или прототип графического интерфейса; • описание архитектуры (UML-диаграммы классов, состояний и последовательностей); • отчёт.
Сдача 1-ой версии	09.07.2021	• Реализация алгоритма; • прототип GUI с частичным функционалом; • тестирование алгоритма; • отчёт.
Сдача 2-ой версии	12.07.2021	• Полностью рабочий GUI и CLI; • реализация взаимодействия с алгоритмом; • тестирование алгоритма; • отчёт.
Финал	14.07.2021	• Внесение правок/устранение недочетов.

2.2. Распределение ролей в бригаде

Участники	Итерация	Роли
Попов Д.С.	Сдача прототипа	Создание прототипа графического интерфейса, проектирование архитектуры.
	Сдача 1-ой версии	Объединение алгоритма с графическим интерфейсом.
	Сдача 2-ой версии	
Рослова Л.С.	Сдача прототипа	Создание UML-диаграмм, проектирование архитектуры.

	Сдача 1-ой версии	Реализация алгоритма.
	Сдача 2-ой версии	
Паутова Ю.В.	Сдача прототипа	Создание UML-диаграмм, проектирование архитектуры.
	Сдача 1-ой версии	Разработка визуализации графа для алгоритма.
	Сдача 2-ой версии	

3. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС

3.1. Итерация 1: сдача прототипа

На рисунках 1-3 представлен внешний вид пользовательского (графического) интерфейса программы-визуализатора алгоритма ЯПД, разрабатываемого в данном проекте. На рисунке 1 представлено стартовое окно программы, которое открывается при запуске приложения. Здесь пользователь может ознакомиться с программой, нажав на кнопку «О программе», или перейти к вводу данных для визуализации алгоритма, нажав на кнопку «Поехали».

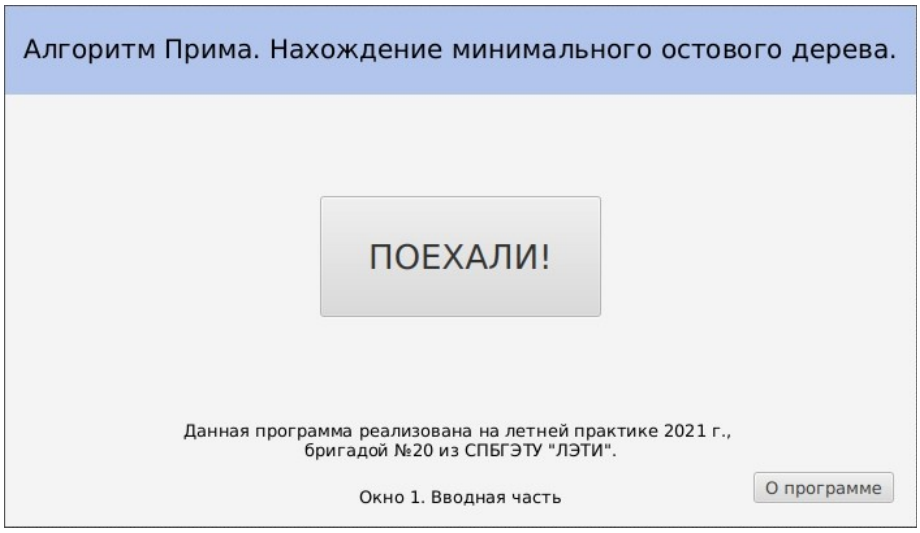

Рисунок 1 – Стартовое окно программы

На рисунке 2 представлено окно ввода данных для алгоритма. Пользователь может ввести данные непосредственно в самом окне или загрузить их из файла, в который заранее занёс данные. Также пользователь может выбрать с какой вершины начать построение минимального остового дерева. После ввода данных пользователь нажимает кнопку «Далее», чтобы перейти к визуальному представлению алгоритма.

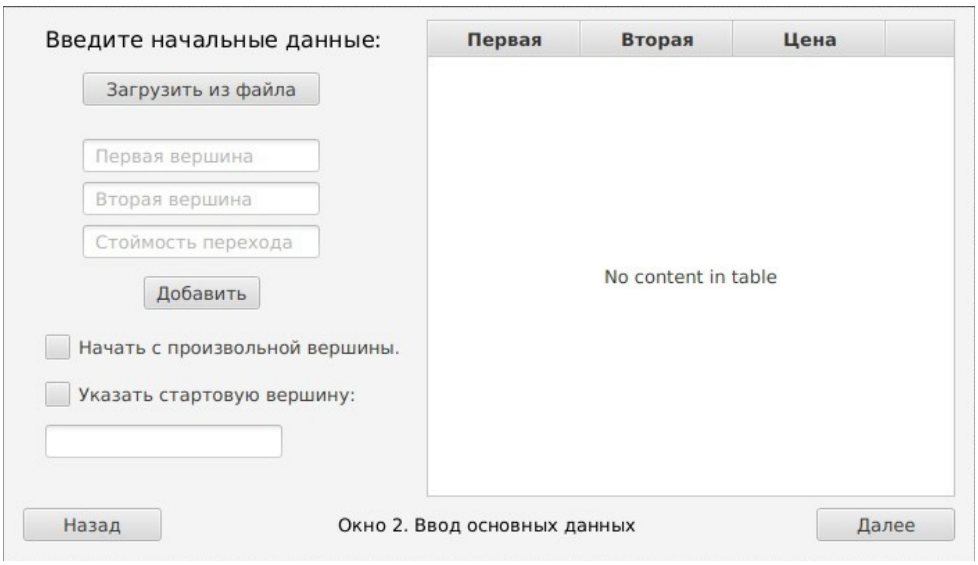

Рисунок 2 – Окно для ввода основных данных

На рисунке 3 представлено окно визуализации. Здесь пользователь может выбирать проходить алгоритм пошагово или до конца, может вернуться к изначальному состоянию графа или же ввести новые данные. Параллельно с визуальным представлением алгоритма в данном окне выводятся логи: какие ребра рассматривались и какое было взято в итоге.

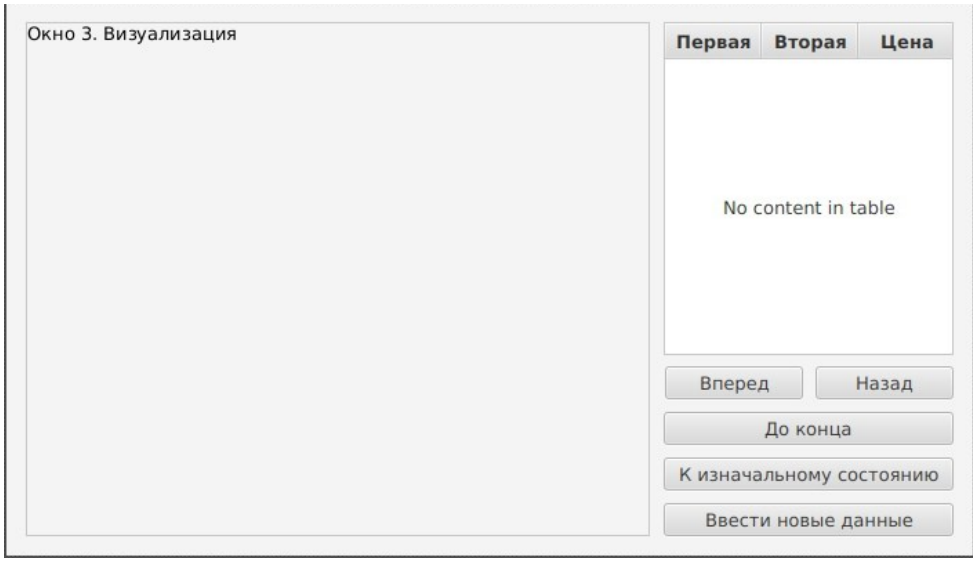

Рисунок 3 – Окно визуализации

3.2. Итерация 2: сдача 1-ой версии

На рисунках 4-6 представлен реализованный функционал пользовательского (графического) интерфейса программы-визуализатора алгоритма ЯПД.

На рисунке 4 представлено окно для ввода данных. В нём был реализован функционал добавления рёбер как через графический интерфейс, так и через передаваемый программе файл. При нажатии кнопки далее происходит инициализация графа и мы переходим на следующее окно.

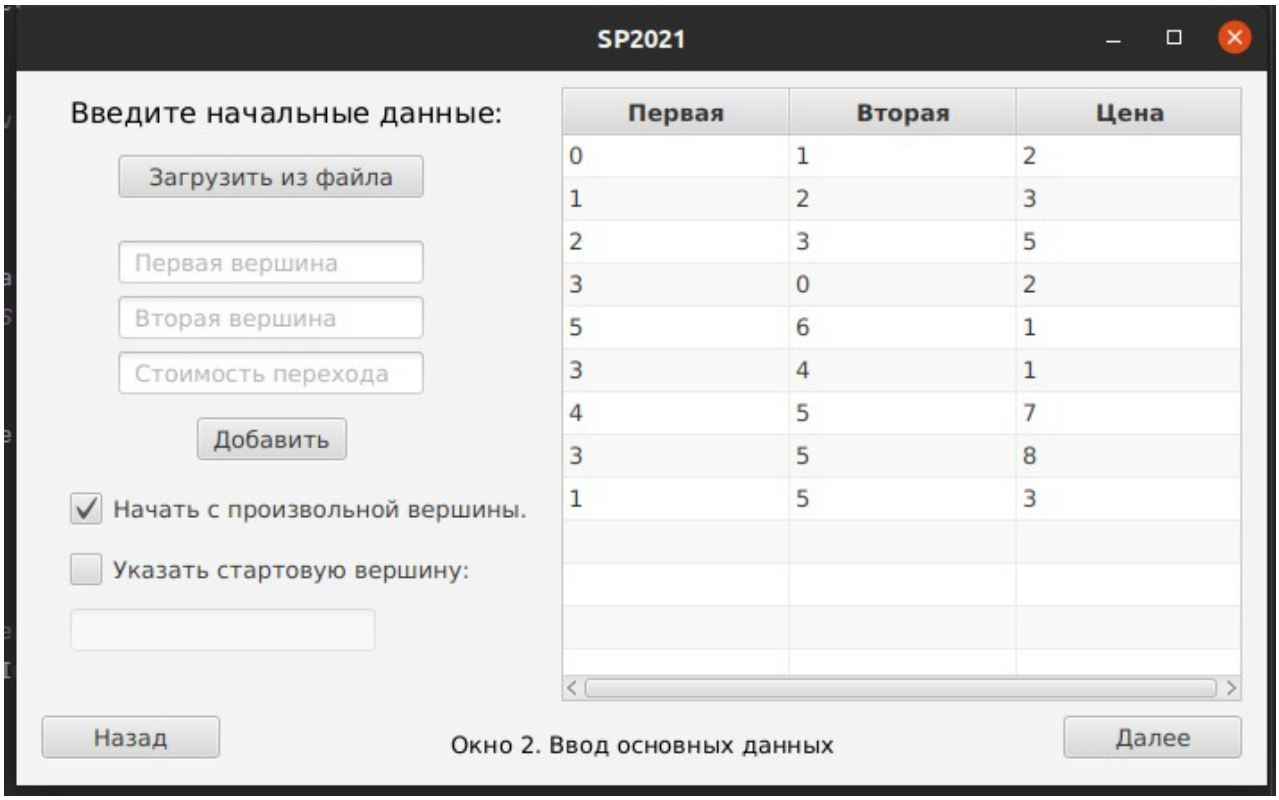

Рисунок 4 - Окно ввода информации

На рисунке 5 представлен граф, который мы инициализировали во 2м окошке программы. Функционал данного окна на данном этапе позволяет двигаться по алгоритму постепенно, откатываться на шаг назад, начать выполнение алгоритма заново или посмотреть сразу вывод отработанной программы.

На рисунке 6 представлен вывод алгоритма на некоторой итерации.

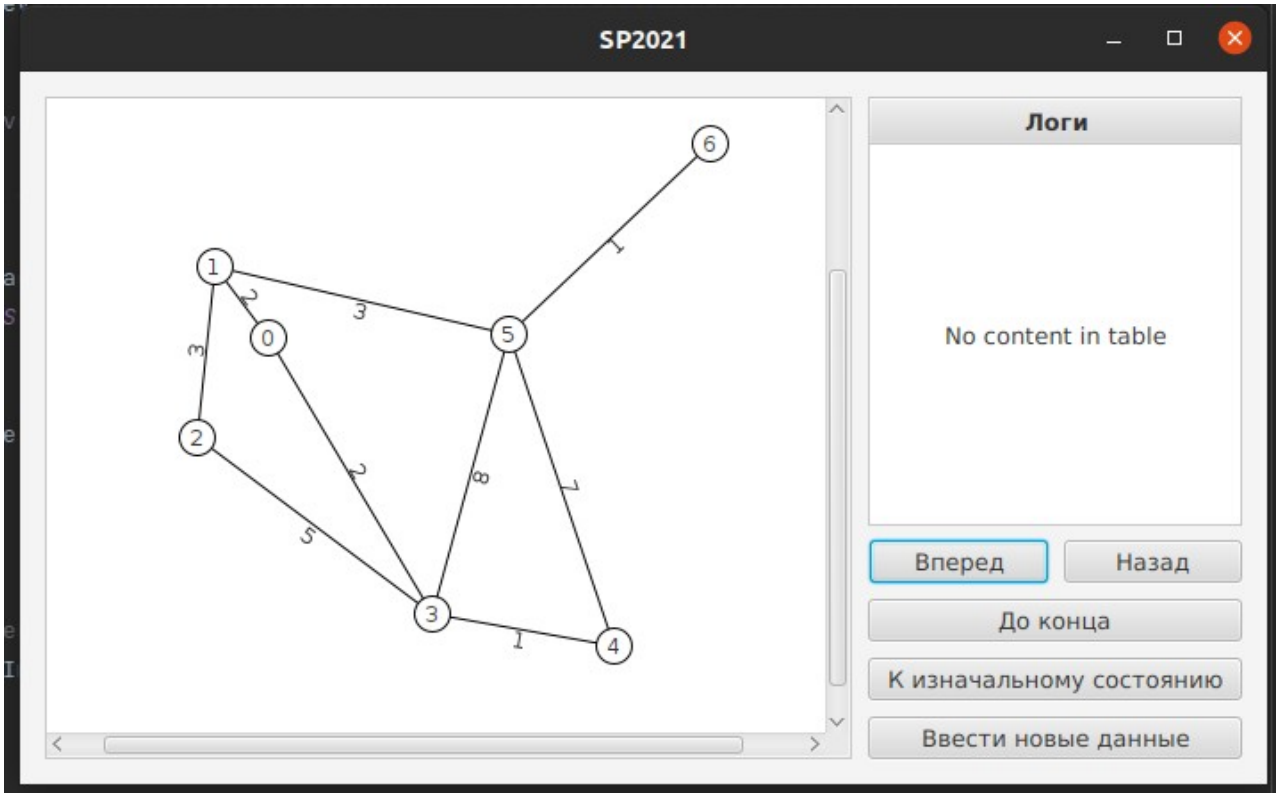

Рисунок 5 — Окно визуализации алгоритма

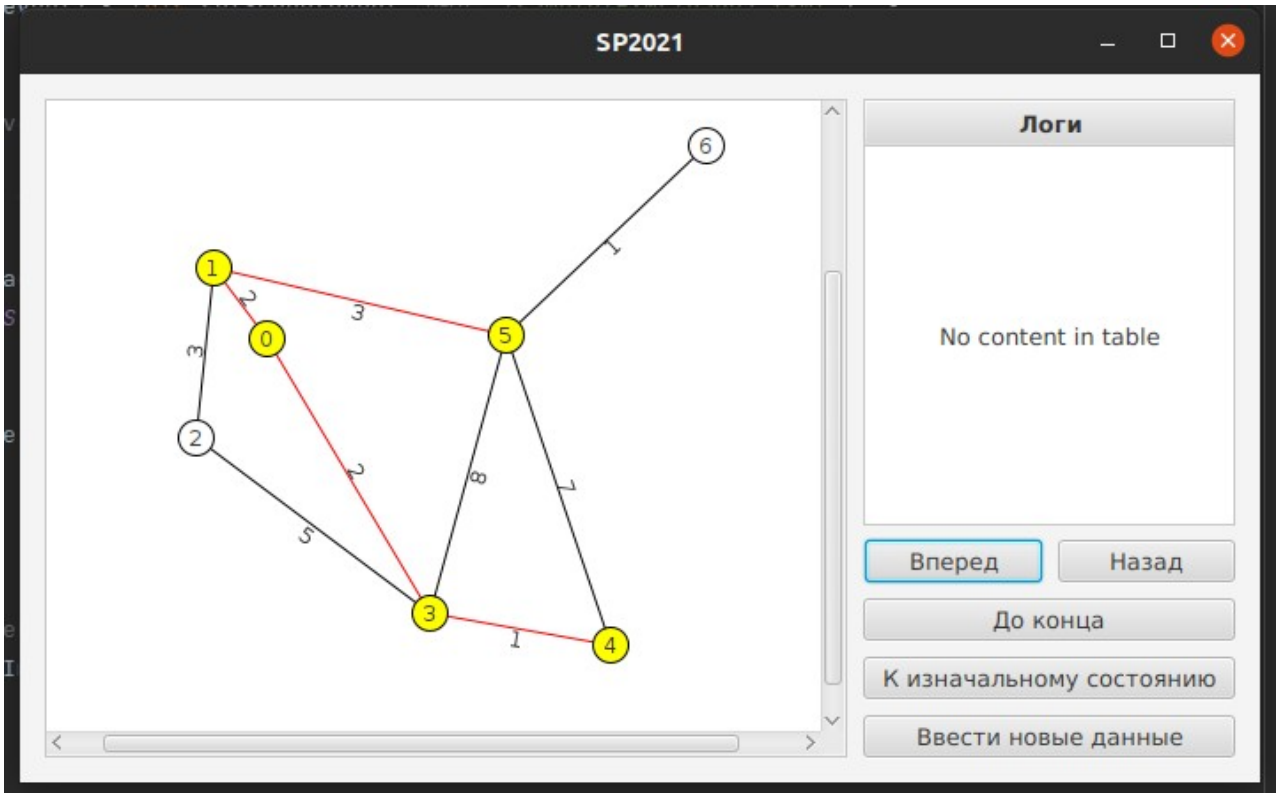

Рисунок 6 — Окно визуализации алгоритма (вывод программы на некоторой итерации)

Для графического представления алгоритма использовалась библиотека JUNG.

Реализация алгоритма Прима

Теорема сегментации: при любой сегментации ребро с наименьшим весом среди поперечных ребер должно принадлежать минимальному остовному дереву.

Придерживаясь этой теоремы был реализован алгоритм:

1. Начиная с 0-й вершины, записываем посещенные вершины
2. Получаем все смежные ребра вершины (вершина на другом конце ребра не должна быть посещена, иначе ребро не является ребром поперечного сечения) и добавляем вес ребра к минимальной куче
3. Берём текущее наименьшее ребро из наименьшей кучи. Согласно теореме о разбиении, это ребро должно быть ребром MST.
4. Установливаем текущую вершину на другую вершину наименьшего ребра и начнинаем повторять процесс 1, 2, 3
5. Когда минимальная куча пуста, построение MST завершено.

Был реализован класс графа *WeightGraph* с полями i*nt edgeNum*, *int vertexNum* и *List<Edge> graph*, отвечающие соответственно за количество рёбер, количество вершин и массив рёбер. В классе также реализован другой класс — *Edge*, который отвечает за хранение рёбер вида вершина — вершина: ребро. Класс графа имеет ряд методов:

- *getVertexNum()* - возвращает количество вершин в графе
- *getEdgeNum()* - возвращает количество рёбер в графе
- *getGraph()* - даёт список рёбер с прилегающими к ним вершинами
- *addEdge()* - добавляет новое ребро в массив
- *getResult()* - возвращает массив чисел — данные для графического представления графа

Для алгоритма был реализован отдельный класс *LazyPrimMST*. Он хранит в себе следующие значения:

- *VisualGraph visGraph* — параметр для визуализации графа
- *WeightGraph graph* — представление графа
- *PriorityQueue<WeightGraph.Edge> minHeap* — минимальная куча, откуда будут стягиваться значения
- *boolean[] marked* — массив для отметки рассмотренных узлов

- *List<WeightGraph.Edge> mst* — массив, куда будет записываться минимальное остовное дерево
- *Integer mstWeight* — вес остова

Алгоритм был реализован в методе *LazyPrimMST(graph)*.

Код описанных классов представлен в приложении А.

3.3. Итерация 3: сдача 2-ой версии

3.4. Финальная версия

4. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

4.1. Описание архитектуры

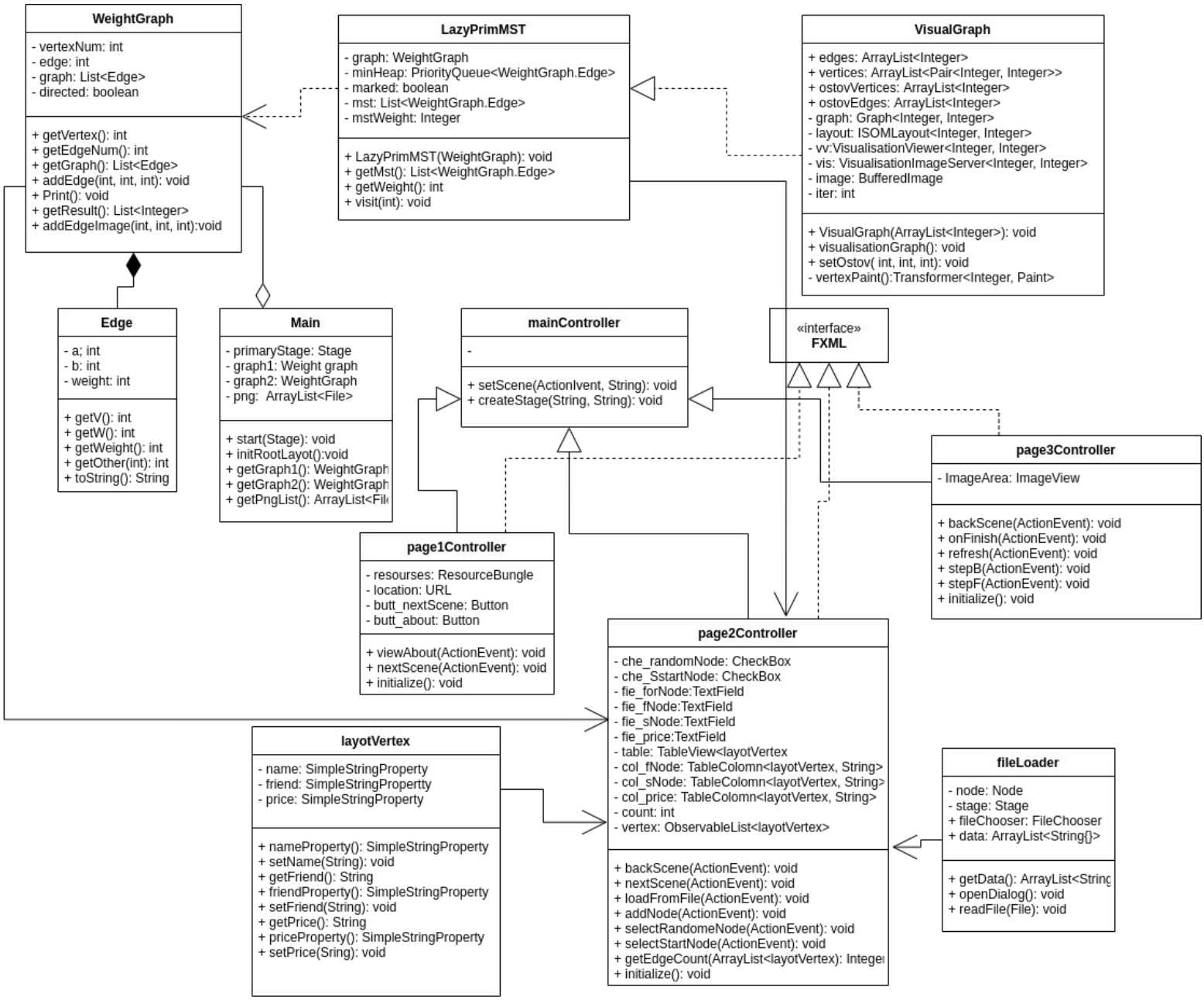

Рисунок 7 – UML-диаграмма классов

Рисунок 7 демонстрирует архитектуру программы на данном этапе реализации проекта. Т.к. инициализация графа для работы программы проходит во 2-м окне приложения, то все основные методы по созданию объектов и присваивания им значений происходят в классе page2Controller, потому что он

реализует загрузку и функционал окна ввода данных. В нём происходит создание считывание значений и создание графа, который передаётся алгоритму.

На рисунке 8 изображена UML-диаграмма состояний. С помощью данной диаграммы описывается алгоритм ЯПД, реализующийся в данном проекте. В начале создается граф по данным, введённым пользователем или считанным из файла, и выбирается вершина, с которой начнется построения минимального остового дерева. После чего запускается алгоритм:

1. У начальной вершины ищется ребро с наименьшей стоимостью. Если ребро не найдено, то в графе одна вершина и алгоритм завершается, иначе добавляем найденное ребро к каркасу.
2. В цикле происходит дополнение остового дерева:
   2.1. Формирование множества рёбер, одна вершина которых помечена, а вторая нет.
   2.2. Выбор из этого множества ребра с наименьшей стоимостью и добавление его к каркасу.
   2.3. Условие завершения цикла: все вершины графа вошли в остовое дерево, то есть были помечены.

Рисунок 8 – UML-диаграмма состояний

На рисунке 9 изображена UML-диаграмма последовательности, которая отображает взаимодействие объектов в динамике. При запуске пользователем программы открывается окно, после чего пользователь вводит данные, которые используются для построения графа. Построенный граф визуализируется и передается графическому интерфейсу для отображения его пользователю. Затем пользователь через интерфейс запускает алгоритм и проходит его пошагово или до конца. Каждая итерация алгоритма визуализируется и отображается пользователю через графический интерфейс.

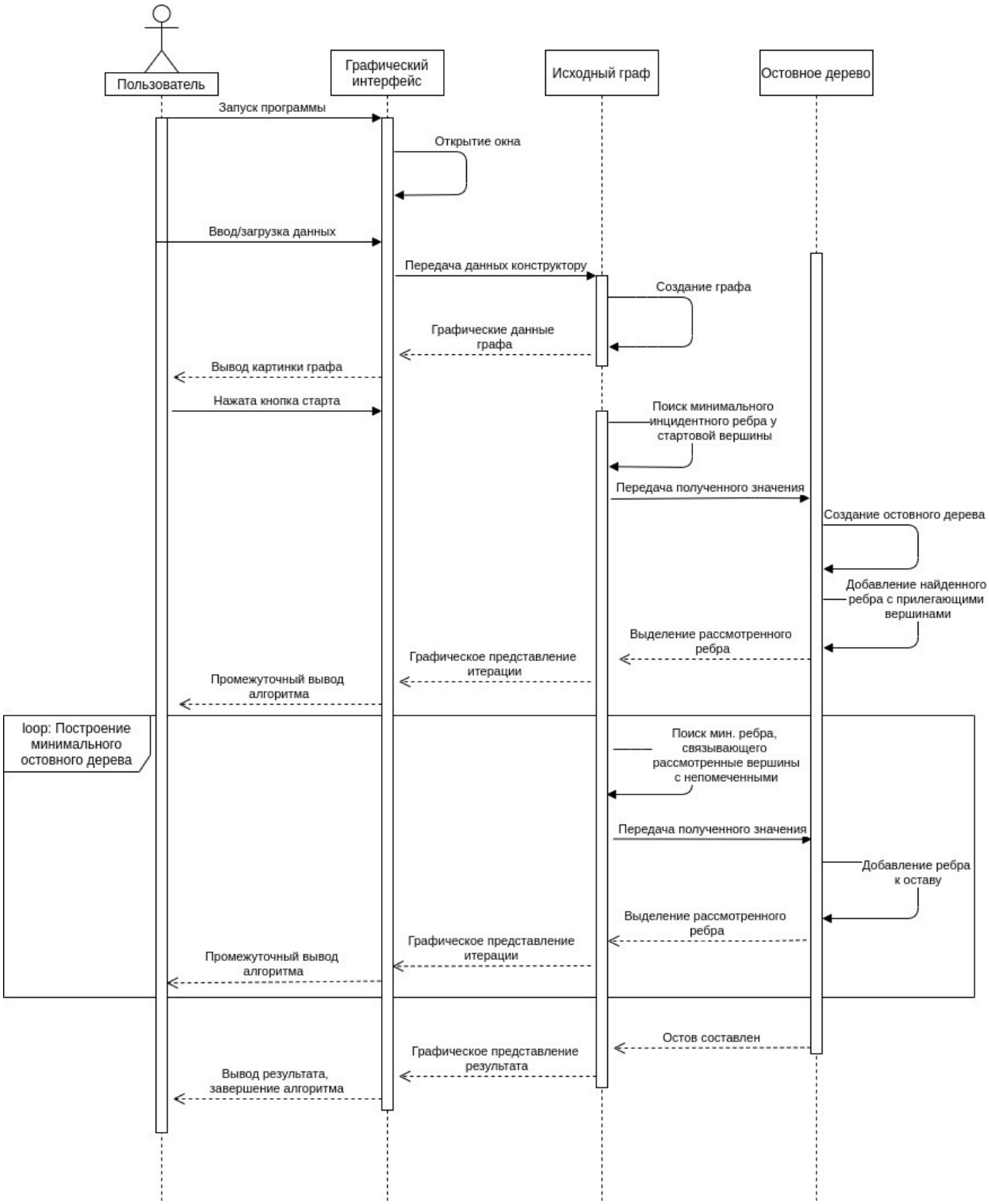

Рисунок 9 – UML-диаграмма последовательности

5. ТЕСТИРОВАНИЕ

5.1. Тестирование 1-ой версии

5.2. Тестирование 2-ой версии

ЗАКЛЮЧЕНИЕ

Разработка проекта учебной практики велась итеративно, то есть создание программы-визуализатора алгоритма ЯПД (Прима) осуществлялось в несколько этапов.

На первом этапе были разработаны архитектура проекта и прототип пользовательского интерфейса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. AL диджитализация бизнеса // evergreens.com.ua. URL: https://evergreens.com.ua/ru/articles/uml-diagrams.html

2. Свободная энциклопедия // wikipedia.org. URL: https://ru.wikipedia.org/wiki/Алгоритм_Прима

3. JUNG: Java Universal Network/Graph Framework // jung.sourceforge.net. URL: http://jung.sourceforge.net/doc/api/index.html

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

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