

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

ОТЧЕТ
по учебной практике
по дисциплине «Построение анализ алгоритмов»
Тема: Визуализация алгоритма Куна.

| | | |
|------------------|-------|----------------|
| Студент гр. 9381 | _____ | Игнашов В. М. |
| Студент гр. 9381 | _____ | Семёнов А. Н. |
| Студент гр. 9381 | _____ | Матвеев А. Н. |
| Преподаватель | _____ | Жангиров Т. Р. |

Санкт-Петербург
2021

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Игнашов В. М. группы 9381

Студент Семёнов А. Н. группы 9381

Студент Матвеев А. Н. группы 9381

Тема практики: Визуализация алгоритма Куна.

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: Алгоритм Куна.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета:

Дата защиты отчета:

Студент гр. 9381

Игнашов В. М.

Студент гр. 9381

Семёнов А. Н.

Студент гр. 9381

Матвеев А. Н.

Преподаватель

Жангиров Т. Р.

АННОТАЦИЯ

Целью практической работы является итеративная разработка программы для визуализации работы алгоритма Куна. Пользователю программы должна быть предоставлена возможность самостоятельно задать входные данные для алгоритма с помощью графического интерфейса, а также возможность пошагового исполнения алгоритма с просмотром промежуточных состояний алгоритма (чередующейся цепи). Разработка осуществляется с использованием языка программирования Java командой из трех человек.

SUMMARY

The purpose of the practical work is the iterative development of the program to visualize the work of Kuhn's algorithm. The user of the program should be given the opportunity to independently specify the input data for the algorithm using the graphical interface, as well as the ability to step-by-step execution of the algorithm with viewing the intermediate states of the algorithm (alternating chain). Development is carried out using the Java programming language by a team of three people.

СОДЕРЖАНИЕ

| | |
|---|----|
| Введение | 5 |
| 1. Требования | 6 |
| 1.1. Проектирование архитектуры | 6 |
| 1.2. Требование к реализации алгоритма | 6 |
| 1.3. Требование к проекту | 6 |
| 2. План разработки | 8 |
| 2.1. План разработки | 8 |
| 2.2. Распределение ролей | 8 |
| 3. Описание архитектуры | 9 |
| 4. Особенности реализации | 12 |
| 4.1. Архитектура программы | 12 |
| 4.2. Описание алгоритма | 13 |
| 4.3. Структуры данных | 14 |
| 4.4. Описание графического интерфейса программы | 17 |
| 5. Тестирование | 19 |
| 5.1. План тестирования программы | 19 |
| 5.2. Случаи для тестирования | 19 |
| 5.3. Результаты тестирования | 25 |
| Заключение | 27 |
| Список использованных источников | 28 |
| Приложение А | 29 |

ВВЕДЕНИЕ

Целью практической работы является итеративная разработка программы для визуализации работы алгоритма Куна для нахождения наибольшего паросочетания в двудольном графе. Пользователю программы должна быть предоставлена возможность самостоятельно задать входные данные для алгоритма с помощью графического интерфейса, а также возможность пошагового исполнения алгоритма с просмотром промежуточных состояний алгоритма. В конечной версии программы должна быть предусмотрена возможность сохранения и загрузки исходных данных для алгоритма из файла.

Разработка осуществляется с использованием языка программирования Java командой из трех человек. Каждому участнику команды назначается роль и выполняемые им задачи (разработка интерфейса, логики алгоритма и тестирование программы). В результате выполнения работы должна быть представлена программа, без ошибок собирающаяся из исходных файлов и протестированная на корректность.

1. ТРЕБОВАНИЯ

Проектирование архитектуры:

- 1) Архитектура должна быть модульной, то есть разделен на разные слои абстракции;
- 2) Описание модулей проекта через UML диаграммы классов;
- 3) Описание работы программы и алгоритма через UML диаграммы состояний;
- 4) Описание работы программы и алгоритма через UML диаграммы последовательности;
- 5) Желательно соблюдать принципы SOLID;
- 6) Желательно использовать паттерны проектирования;

Требование к реализации алгоритма:

- 1) Алгоритм должен реализован так, чтобы можно было использовать любой тип данных. (Например через generic классы);
- 2) Алгоритм должен поддерживать возможность включения промежуточных выводов и пошагового выполнения.

Требование к проекту:

- 1) Возможность запуска через GUI и по желанию CLI (в данном случае достаточно вывода промежуточных выводов);
- 2) Загрузка данных из файла или ввод через интерфейс;
- 3) GUI должен содержать интерфейс управления работой алгоритма, визуализацию алгоритма, окно с логами работы;
- 4) Должна быть возможность запустить алгоритма заново на новых данных без перезапуска программы;
- 5) Должна быть возможность выполнить один шаг алгоритма, либо завершить его до конца. В данном случае должны быть автоматически продемонстрированы все шаги;
- 6) Должна быть возможность вернуться на один шаг назад;

7) Должна быть возможность сбросить алгоритма в исходное состояние.

На рисунке 1 представлен макет интерфейса программы:

2. ПЛАН РАЗРАБОТКИ

2.1. План разработки

- Составление спецификации
- Распределение ролей
- Разработка прототипа пользовательского интерфейса
- Возможность построения графа в пользовательском интерфейсе
- Просмотр начального и конечного состояний алгоритма
- Чтение/сохранение графа в файл
- Возможность пошагового исполнения алгоритма
- Возможность возврата алгоритма к предыдущим состояниям
- Тестирование корректности работы программы
- Реализация дополнительного функционала

2.2. Распределение ролей.

Семёнов: Разработка/отладка алгоритма и структур данных + организация связи интерфейса с логикой;

Игнашов: Разработка и отладка графического интерфейса + конструктивная отрисовка графа;

Матвеев: Юнит-тестирование + логирование/организация файлового ввода-вывода + участие в связи интерфейса с логикой;

3. ОПИСАНИЕ АРХИТЕКТУРЫ.

Ниже представлены UML-диаграммы классов, состояний и последовательностей.

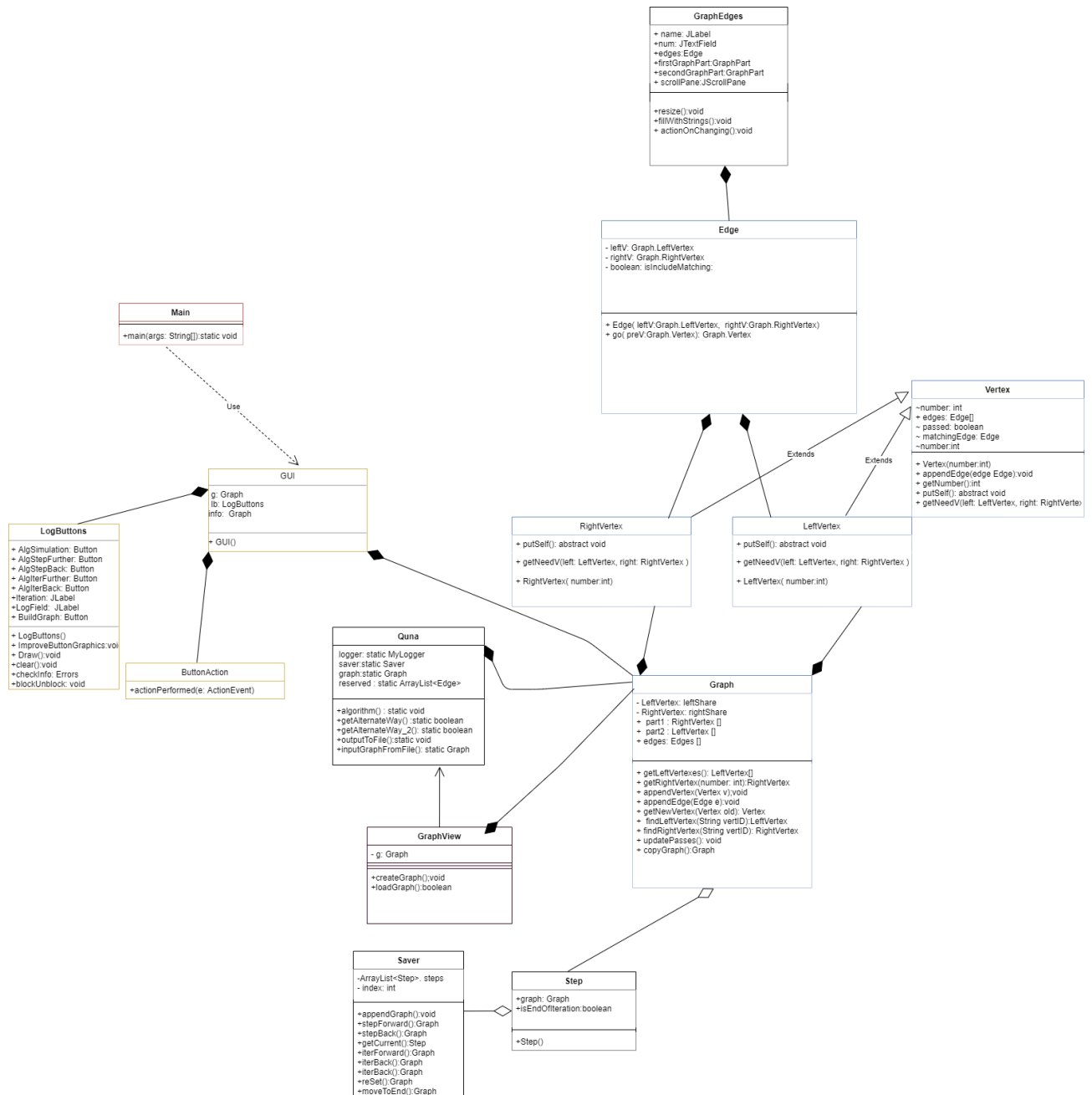


Рис. 1. Диаграмма классов.

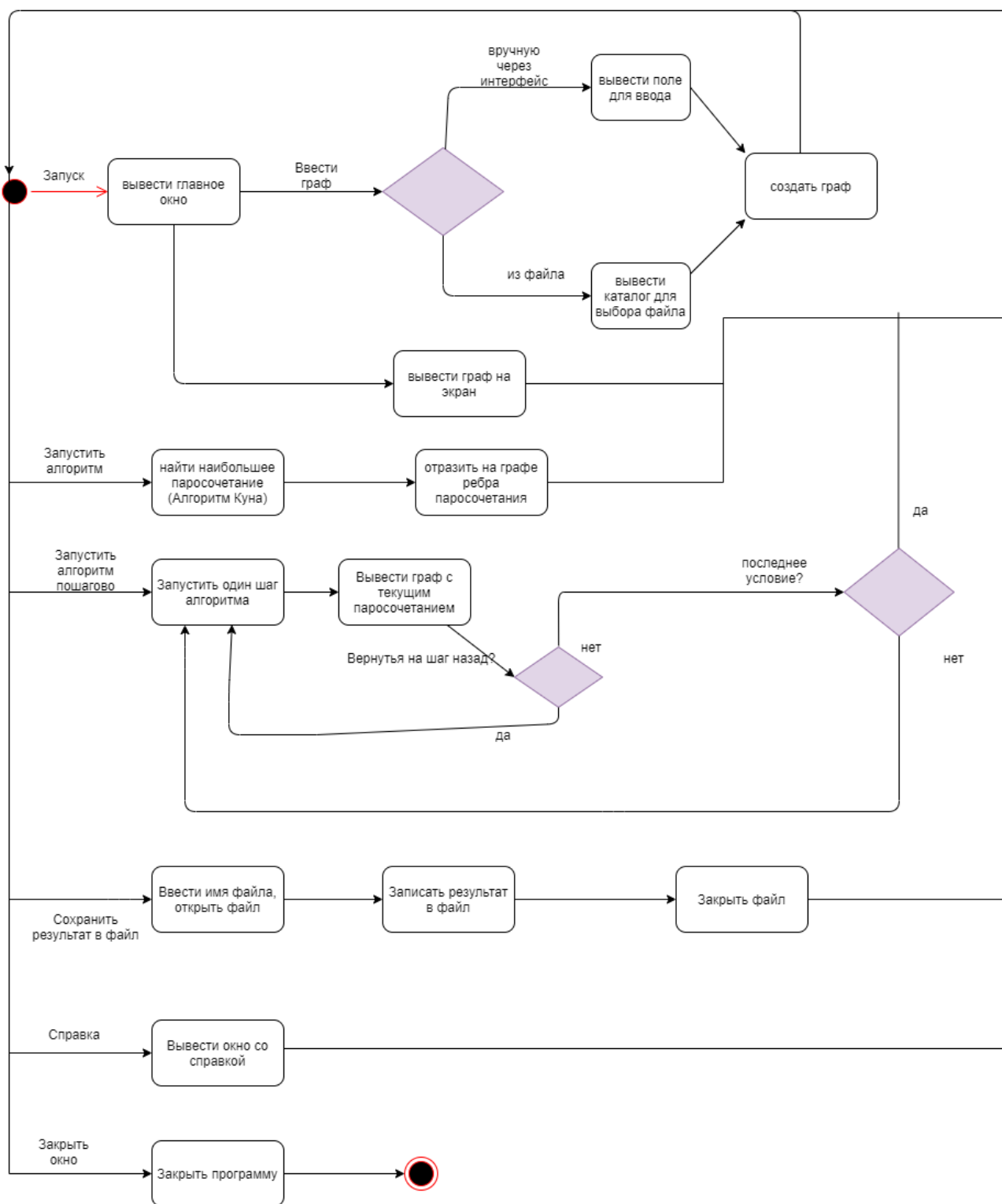


Рис. 2. Диаграмма состояний.

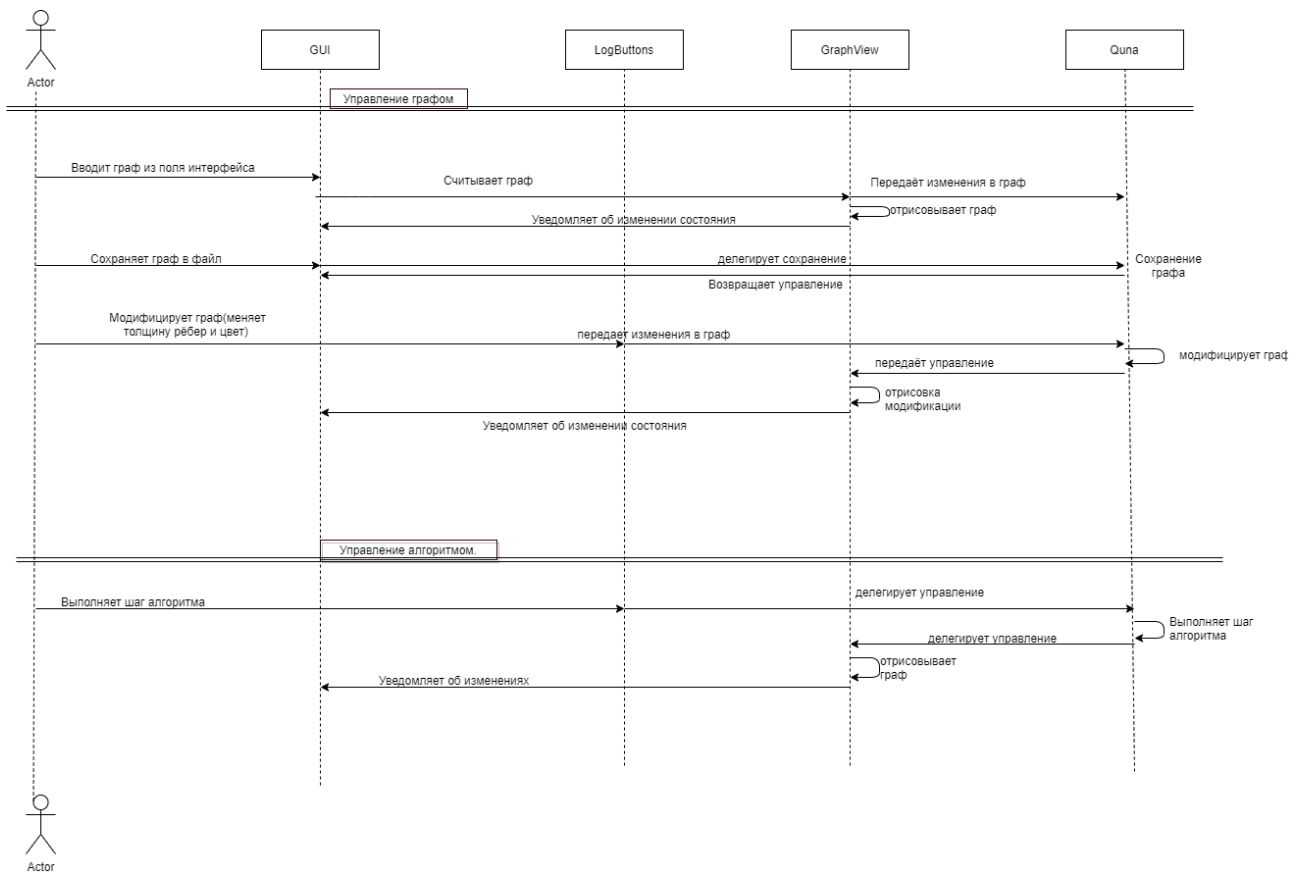


Рис. 3. Диаграмма последовательностей.

4. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

4.1. Архитектура программы.

Main - основной класс. Его одноимённый метод создает экземпляр GUI и делает его видимым, а также включает инструкции.

GUI - класс графического интерфейса. Отвечает за отрисовку слоёв, текстовых полей, расположение графических объектов друг относительно друга.

Errors - класс-перечисление. Содержит виды ошибок.

GraphEdges - класс, который отвечает за слой, в котором пользователь вводит рёбра с интерфейса, содержит информацию о графе. (Набор рёбер). Также отвечает за их построение на интерфейсе.

Edge - класс ребра.

GraphPart - класс, отвечающий за доли графа и содержащий информацию о них. Визуализирует панели ввода вершин обеих долей.

GraphInfo - класс, содержащий информацию о графе (в частности части интерфейса для левой доли, правой доли рёбра и методы для работы с ними).

LogButtons - класс, содержащий кнопки, с каждой из которых связано конкретное действие пользователя. Например, когда пользователь нажимает на кнопку build graph, строится и отрисовывается граф.

GraphView - отображение графа. Имеет информацию о координатах точек и рёбер будущего графа (Для рёбер есть координаты начала и конца). Она хранится в виде массивов. Отвечает за отрисовку и разметку графа на интерфейсе.

Graph - класс, содержащий информацию о графе, но уже с точки зрения логики и работы алгоритма. Является основной структурой данных, обрабатываемой алгоритмом.

MyLogger - класс-логгер, который производит запись логов в специальную переменную.

Step - класс-снимок графа. Представляет состояние графа в разные моменты работы алгоритма и содержит информацию о том, является ли текущее состояние графа концом итерации.

Saver - класс, который содержит последовательность состояний графа в разные моменты работы алгоритма и позволяет делать шаг вперёд и шаг назад, итерацию вперёд, итерацию назад.

Quna - класс, отвечающий за работу алгоритма. Позволяет считывать граф с файла, записывать в файл, производить алгоритм, заполняя ленту состояний графа и записывая промежуточные выводы в логгер.

Vertex - класс вершины.

LeftVertex - класс вершины из левой доли.

RightVertex - класс вершины из правой доли.

QunaTests - класс, производящий юнит-тестирование алгоритма на разных входных данных.

4.2. Описание алгоритма.

Цепью длины k назовём некоторый простой путь (т.е. не содержащий повторяющихся вершин или рёбер), содержащий ровно k рёбер.

Чередующейся цепью относительно некоторого паросочетания назовём простой путь длины k в которой рёбра поочередно принадлежат/не принадлежат паросочетанию.

Увеличивающей цепью относительно некоторого паросочетания назовём чередующуюся цепь, у которой начальная и конечная вершины не принадлежат паросочетанию.

С помощью увеличивающейся цепи увеличивать паросочетание на единицу. Можно взять такой путь и провести чередование — убрать из паросочетания все рёбра, принадлежащие цепи, и, наоборот, добавить все остальные. Всего в увеличивающей цепи нечетное число рёбер, а первое и последнее были не в паросочетании. Значит, мощность паросочетания увеличилась ровно на единицу.

Алгоритм Куна: поиск увеличивающейся цепи, пока это возможно, и проведение чередование в ней. Увеличивающие цепи удобны тем, что их легко искать: можно просто запустить поиск пути из произвольной свободной вершины из левой доли в какую-нибудь свободную вершину правой доли в том же графе, но в котором из правой доли можно идти только по рёбрам паросочетания (то есть у вершин правой доли будет либо одно ребро, либо ноль). Это можно делать как угодно, однако устоялась эффективная реализация в виде поиска в глубину.

Сложность алгоритма: $O(E \cdot V)$. E - количество ребер. V - количество вершин.

4.3. Структуры данных

4.3.1. Vertex - абстрактный класс вершины, являющийся родителем классов вершин из правой и левой долей.

Поля:

- protected String name - идентификатор вершины.
- public ArrayList<Edge> edges - список рёбер, инцидентных данной вершине;
- protected boolean passed - флаг посещения вершины в построении пути;
- protected Edge matchingEdge; - ссылка на ребро, инцидентное данной вершине;
- protected boolean isResearchingNow - флаг того, исследуется ли вершина сейчас.

Методы:

- public Vertex(String name) - конструктор класса;
- public Vertex(Vertex toBeCopied) - конструктор копирования класса;
- геттеры и сеттеры для полей класса;
- public void appendEdge(Edge edge) - добавление ребра к вершине;
- public abstract void putSelf(); - кладёт экземпляр класса в граф. Будет реализован в классах наследниках;

- `public abstract Vertex getNeedV(LeftVertex left, RightVertex right)` - возвращает ту долю графа, которая расположена на противоположной стороне от текущей вершины.
- `class LeftVertex` и `class RightVertex` наследуют класс вершины и реализуют по своему `putSelf`, и `getNeedV`.

4.3.2. Edge.

Поля:

- `private Graph.LeftVertex leftV` - левая вершина ребра;
- `private Graph.RightVertex rightV` - правая вершина ребра;
- `private boolean includeMatching` - флаг входит ли ребро в паросочетание;
- `private boolean isResearchingNow` - исследуется ли ребро сейчас.

Методы:

- Геттеры и сеттеры для полей класса;
- Конструктор класса;
- Конструктор копирования;
- `public Graph.Vertex go(Graph.Vertex preV, boolean isFurther)` - переходит по ребру из `preV` и возвращает вершину, куда мы перешли;
- `public boolean isIncludeMatching()` - возвращает значение, входит ли вершина в паросочетание.
- `public void doIncludeInMatching()` - включить ребро в паросочетание;
- `public void doExcludeFromMatching()` - выключить ребро в паросочетания.

4.3.3. Graph.

Поля:

- `private ArrayList<LeftVertex> leftShare` - список вершин левой доли;
- `private ArrayList<RightVertex> rightShare` - список вершин правой доли;
- `private ArrayList<Edge> graphEdges` - список рёбер графа;
- `static StringBuilder log` - переменная, в которую последовательно записываются логи;
- `private ArrayList<Edge> maxMatching` - список рёбер максимального паросочетания.

Методы:

- конструктор класса;
- `public void appendVertex(Vertex v)` - добавляет вершину в граф;
- `public void appendEdge(Edge e)` - добавляет ребро в граф;
- `public Vertex getNewVertex(Vertex old)` - возвращает вершину графа с названием `old`;
- `public LeftVertex findLeftVertex(String vertID)` - возвращает вершину из левой доли с таким названием или `null` если такой вершины нет; Для вершины из правой доли существует аналогичный метод;
- геттеры и сеттеры;
- `public Edge findEdges(String leftName, String rightName)` - возвращает ссылку на заданное ребро в графе, если таковое имеется; в противном случае возвращает `null`;
- `public void updatePasses()` - сбросить данные о пройденных вершинах на текущей итерации;
- `public ArrayList<Edge> getResult()` - возвращает список с наибольшим паросочетанием, если он непуст, иначе `null`;
- `public Graph copyGraph()` - метод, который возвращает копию графа.

4.3.4. Quna.

Поля:

- static MyLogger logger - экземпляр класса-логгера;
- static Saver saver - экземпляр класса-хранителя состояний;
- static Graph graph - экземпляр графа;
- static ArrayList<Edge> reserved - зарезервированное поле, необходимое для тестирования.

Методы:

- public static void algorithm() - класс, запускающий алгоритм;
- static boolean getAlternateWay(Graph.Vertex vertex, ArrayList<Edge> list) - одна из взаимно-рекурсивных функций, необходимая для поиска в глубину;
- static boolean getAlternateWay_2(Graph.Vertex vertex, ArrayList<Edge> list) - другая из взаимно-рекурсивных функций, необходимая для поиска в глубину;
- static void outputToFile(String pathTo) - вывод в файл графа;
- static Graph inputGraphFromFile (String pathFrom) - ввод графа из файла.

4.4. Описание графического интерфейса программы.

Для программы был разработан графический интерфейс, который позволяет пользователю выполнять следующие действия:

- Загрузка графа из файла;
- Сохранение графа в файл;
- Ввод графа через удобный интерфейс;
- Отрисовка графа (Build Graph);
- Перезапуск алгоритма (Reset);
- Запуск анимированного автоматического выполнения алгоритма (Algorithm Simulation);
- Возможность пошагового выполнения алгоритма по шагам и итерациям (Step further, Step back, Iteration further, Iteration back);
- Получение информации об алгоритме и справочной информации;

- Выход из программы.

Все аварийные ситуации, создаваемые пользователем обрабатываются на этапе ввода из файла и интерфейса. При визуализации алгоритма ребра могут менять цвет и толщину, вершины меняют цвета. Так, чтобы показать, что вершина просмотрена на данной итерации, она отмечается зелёным. Чтобы показать, что вершина рассматривается сейчас, она выделяется красным. Чтобы показать что ребро находится в паросочетании, оно выделяется синим. Чтобы показать, что ребро рассматривается в данный момент, оно утолщается.

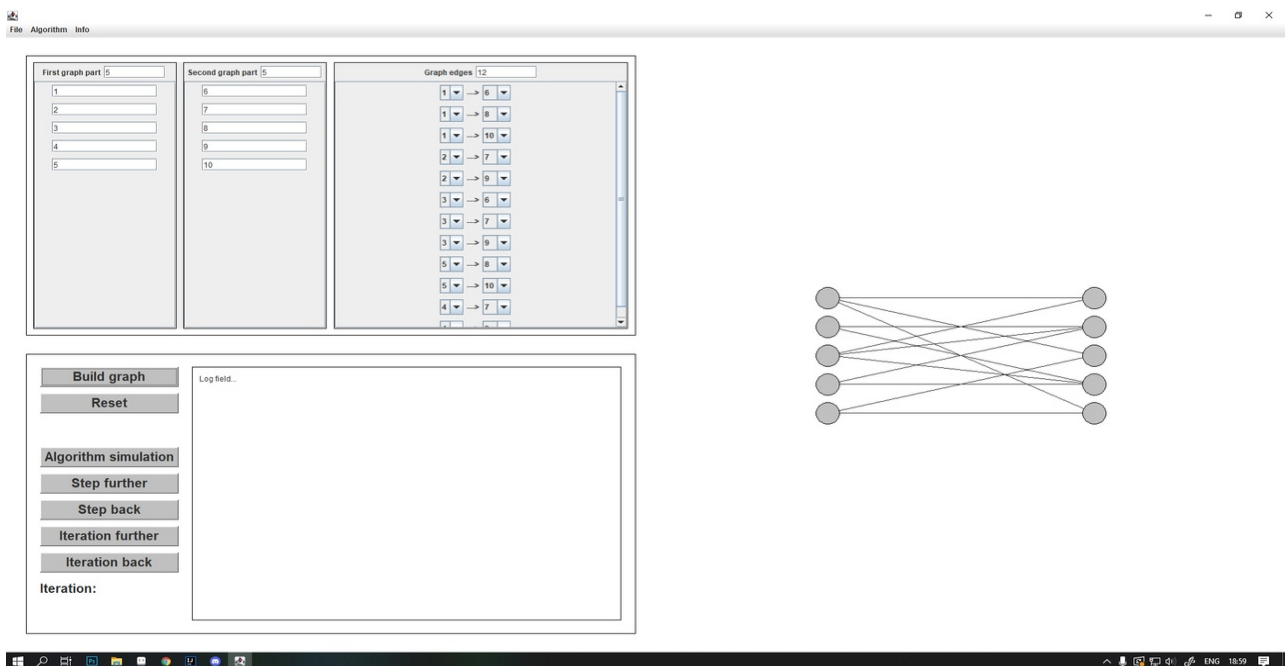


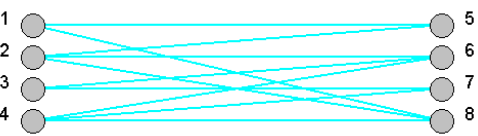
Рис. 4. Текущее состояние интерфейса и отрисовка графа.

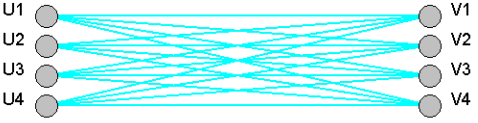
5. ТЕСТИРОВАНИЕ

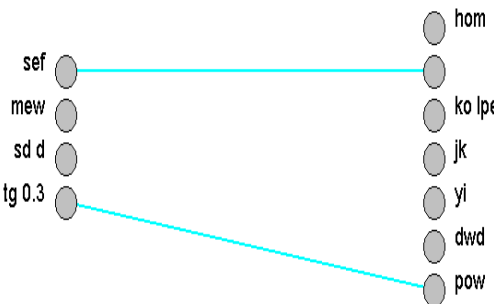
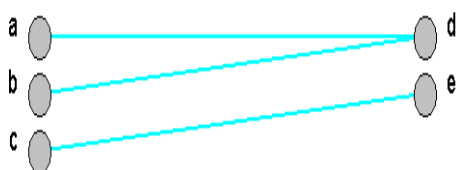
4.1. План тестирования программы.

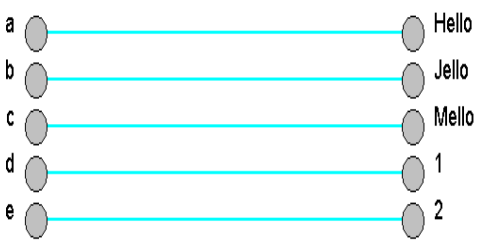
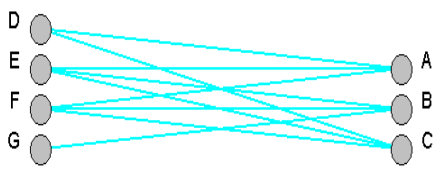
- 1) Объект тестирования: Объектом тестирования является программа для визуализации работы алгоритма Куна.
- 2) Тестируемый функционал: Требуется протестировать метод `algorithm()` класса `Quna`. В этом методе происходит выполнение алгоритма.
- 3) Способ тестирования: Тестирование будет проводиться на модульном уровне при помощи фреймворка автоматического тестирования JUnit.
- 4) Условие завершения тестирования: Тестирование считается успешно завершенным, если все тесты выполнены без ошибок. В противном случае программа возвращается на доработку.

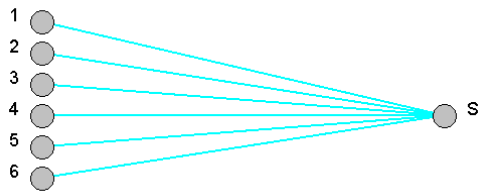
4.2. Случаи для тестирования.

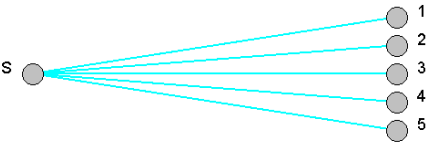

| № | Особенности входных данных | Входные данные | Результат |
|---|--|---|------------------------------|
| 1 |  <p>Произвольный не разреженный граф с одинаковым количеством вершин в обеих долях (без изолированных вершин)</p> | 4 1 2 3 4 4 5 6 7 8 10 1 5 1 8 2 5 2 6 2 8 3 6 3 | 1--5 2--8 3--7 4--6 |

| | | | |
|---|---|--|--------------------------------------|
| | | 7 4 6 4 7 4 8 | |
| 2 | Нуль-граф (3 вершины в каждой доле) | 3 a b c 3 d e f 0 | |
| 3 | Полный 4-вершинный граф Разветвлённый обход в глубину  | 4 u1 u2 u3 u4 4 v1 v2 v3 v4 16 u1 v1 u1 v2 u1 v3 u1 v4 u2 v1 u2 v2 u2 v3 u2 | u1--v4 u2--v3 u3--v2 u4--v1 |

| | | | |
|---|--|--|----------------------|
| | | v4 u3 v1 u3 v2 u3 v3 u3 v4 u4 v1 u4 v2 u4 v3 u4 v4 | |
| 4 | Разреженный граф с большим количеством изолированных вершин  | 4 sef mew sd d tg 0.3 7 hom — ko lpe jk yi dwd pow 2 sef — tg 0.3 pow | sef-- tg 0.3--pow |
| 5 |  | 3 a b c 2 d e | a--d c--e |

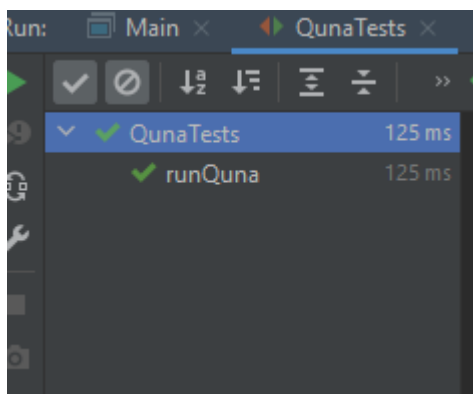
| | | | |
|----|---|---|--|
| | | 3 a d b d c e | |
| 6. |  | 5 a b c d e 5 Hello Jello Mello 1 2 5 a Hello b Jello c Mello d 1 e 2 | a--Hello b--Jello c--Mello d--1 e--2 |
| 7. | <p>Произвольный двудольный граф. Несколько ветвей обхода в глубину.</p>  | 4 D E F G 3 A B C 9 D | D--C E--B F--A |

| | | | |
|----|---|--|------|
| | | A D C E A E B E C F A F B F C G B | |
| 8. | <p>Звезда (в ней алгоритм отработает за количество итераций, равное количеству рёбер)</p>  | 6 1 2 3 4 5 6 1 S 6 1 S 2 S 3 S 4 S 5 S 6 S | 1--S |

| | | | |
|-----|---|---|--------------|
| 9. |  <p>Звезда в другую сторону: в ней алгоритм отработает за одну итерацию.</p> | 1 S 5 1 2 3 4 5 5 S 1 S 2 S 3 S 4 S 5 | S--1 |
| 10. |  | 2 a b 2 c d 2 a d b c | a--d b--c |
| 11. | Пустой граф (ни ребер ни вершин) | 0 0 0 | null |

4.3. Результаты тестирования.

4.3.1. Скриншот запуска unit-теста:



В таблице представлены все тест-кейсы, необходимые для полной проверки алгоритма. В том числе отражены проверки частных случаев алгоритма. (нуль граф, пустой граф). В случае пустого графа в массиве максимального паросочетания будет null. Но подобный ввод запрещён и отлавливается на этапе ввода, так же как и другие виды некорректного ввода. Поэтому программа всегда завершается корректно.

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были изучены основы языка Java, приобретён опыт в визуализации алгоритмов и построения интерфейсов, при помощи фреймворка Swing. Написана, отлажена и протестирована программа с графическим интерфейсом, которая позволяет визуализировать алгоритм Куна. Программа полностью соответствует поставленным в начале разработки требованиям: у пользователя есть возможность задать входные данные для алгоритма с помощью графического интерфейса, а также возможность пошагового исполнения алгоритма с просмотром его промежуточных состояний.

Разработка осуществлялась при активном участии всех членов команды с предварительным распределением ролей в проекте. Во время разработки были получены практические знания о языке программирования Java, фреймворке для разработки пользовательских интерфейсов Swing и фреймворке для юнит-тестирования JUnit. Также были получены навыки проектирования программ с использованием представления внутренних процессов программы в виде диаграмм и навыки командной разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Java Platform, Standard Edition 8 API Specification // Oracle Help Center. URL: <https://docs.oracle.com/javase/8/docs/api/overview-summary.html> (дата обращения: 07.07.2021).
2. Java. Базовый курс // Stepik. URL: <https://stepik.org/course/187/info> (дата обращения: 05.07.2021).
3. JavaFX Reference Documentation // JavaFX. URL: <https://openjfx.io/> (дата обращения: 07.07.2021).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД АЛГОРИТМА

```
import java.io.*;
import java.util.Scanner;
import java.util.ArrayList;

public class Quna { // если
    static MyLogger logger = new MyLogger();
    static Saver saver;
    static Graph graph;
    static ArrayList<Edge> reserved;

    public static void algorithm(){
        saver = new Saver();

        saver.appendGraph(graph, true);

        for (Graph.Vertex v : graph.getLeftVertexes()) {
            if (v.getMatchingEdge() == null) {
                ArrayList<Edge> list = new ArrayList<Edge>();
                v.setResearchingNow();
                if (getAlternateWay(v, list)) {
                    for (int i = 0; i < list.size() / 2; i++) {
                        list.get(i * 2 + 1).doExcludeFromMatching();
                    }
                    for (int i = 0; i < list.size() / 2 + 1; i++) {
                        list.get(i * 2).doIncludeInMatching();
                    }

                    saver.appendGraph(graph, true);

                }
                graph.updatePasses();
                saver.appendGraph(graph, true);
            }
        }
        logger.writeLog("Max matching is found. End of algorithm\n");
        System.out.print(logger.getLogMes());
    }
}
```

```

    }

    static boolean getAlternateWay(Graph.Vertex vertex, ArrayList<Edge>
list) { // поиск в глубину слева направо
        if (vertex.edges.size() == 0) saver.appendGraph(graph, true);
        else saver.appendGraph(graph, false);
        if (vertex.getPassed()) return false;
        vertex.setPassTrue();
        for (Edge e : vertex.edges) {
            if (!e.isIncludeMatching()) {
                list.add(e);
                //saver.appendGraph(graph, false);
                logger.writeLog("Adding edge " + e + " to current
way\n");

                Graph.Vertex nextV = e.go(vertex, true);
                if (getAlternateWay_2(nextV, list)) return true;
                e.go(nextV, false);
                list.remove(list.size() - 1);
                if (list.size() == 0) saver.appendGraph(graph, true);
                else saver.appendGraph(graph, false);
                logger.writeLog("Delete " + e + " from current way\n"
);
            }
        }
        return false;
    }
}

```

```

    static boolean getAlternateWay_2(Graph.Vertex vertex,
ArrayList<Edge> list) { // поиск в глубину справа налево
        saver.appendGraph(graph, false);
        if (vertex.getPassed()) return false;
        vertex.setPassTrue();
        if (vertex.getMatchingEdge() == null) return true;
        list.add(vertex.getMatchingEdge());
        logger.writeLog("Adding edge " + vertex.getMatchingEdge() + "
to current way\n");

        Graph.Vertex nextV = vertex.getMatchingEdge().go(vertex, true);
        if (getAlternateWay(nextV, list)) return true;
    }
}

```

```
vertex.getMatchingEdge().go(nextV, false);
list.remove(list.size() - 1);
saver.appendGraph(graph, false);
    logger.writeLog("Delete " + vertex.getMatchingEdge() + " from
current way\n");
    return false;
}
```