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

ОТЧЕТ

по учебной практике

ТЕМА: «Алгоритм Куна»

Студент гр. 9383 _____ Орлов Д.С.

Студентка гр. 9383 _____ Карпекина А.А

Студентка гр. 9383 _____ Лихашва А.Д.

Руководитель _____ Жангиров Т.Р.

Санкт-Петербург

2021

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Орлов Д.С. группы 9383

Студентка Карпекина А.А. группы 9383

Студентка Лихашва А.Д. группы 9383

Тема практики: Алгоритм Куна

Задание на практику:

Командная разработка визуализации алгоритма на языке Java с графическим интерфейсом.

Алгоритм: Алгоритм Куна.

Дата сдачи отчёта:

Дата защиты отчёта:

Студент гр. 9383	_____	Орлов Д.С.
Студентка гр. 9383	_____	Карпекина А. А.
Студентка гр. 9383	_____	Лихашва А.Д.
Руководитель	_____	Жангиров Т.Р.

АННОТАЦИЯ

Целью учебной практики является разработка графического приложения для нахождения наибольшего паросочетания в двудольном графе при помощи алгоритма Куна.

Программа разрабатывается на языке Java, командой из трёх человек, каждый из которых имеет определённую специализацию.

SUMMARY

The purpose of the training practice is to develop a graphical application for finding the largest match in a bipartite graph using the Kuhn algorithm.

The program is developed in the Java language, by a team of three people, each of whom has a specific specialization.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной практической работы является разработка графического приложения, выполняющего визуализацию работы алгоритма Куна, то есть нахождение наибольшего паросочетания в двудольном графе.
Пользователю программы должна быть предоставлена возможность самостоятельно задать входные данные для алгоритма с помощью графического интерфейса. Результат работы алгоритма должен иметь графическое отображение. Должна быть предоставлена возможность просмотра итогового результата алгоритма и просмотра хода его исполнения по шагам.

Разработка осуществляется на языке Java, командой из трёх человек, каждый из которых имеет определённую специализацию.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные требования к программе

Программа представляет собой визуализацию алгоритма Куна, нахождения наибольшего паросочетания в двудольном графе.

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

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

- Распределение ролей

  Срок выполнения: 02.07.2021

- Сдача вводного задания

  Срок выполнения: 05.07.2021

- Выполнение первой итерации. Эскиз GUI и описание архитектуры.

  Срок выполнения: 06.07.2021

- Выполнение второй итерации. Прототип GUI с реализацией частичного функционала, а также реализация самого алгоритма.
  Срок выполнения: 09.07.2021

- Выполнение третьей итерации. Полностью рабочий GUI, реализация взаимодействия с алгоритмом.

  Срок выполнения: 12.07.2021

- Финальные поправки или устранение недочетов.

  Срок выполнения: 14.07.2021

2.2. Распределение ролей в бригаде

Орлов Д.С.– реализация GUI, реализация взаимодействия алгоритма с GUI.

Карпекина А.А. – реализация алгоритма, составление прототипа GUI, тестирование.

Лихашва А.Д. – реализация алгоритма, визуализация алгоритма, составление UML.

3. ОПИСАНИЕ ПРОЕКТА ЧЕРЕЗ UML-ДИАГРАММЫ

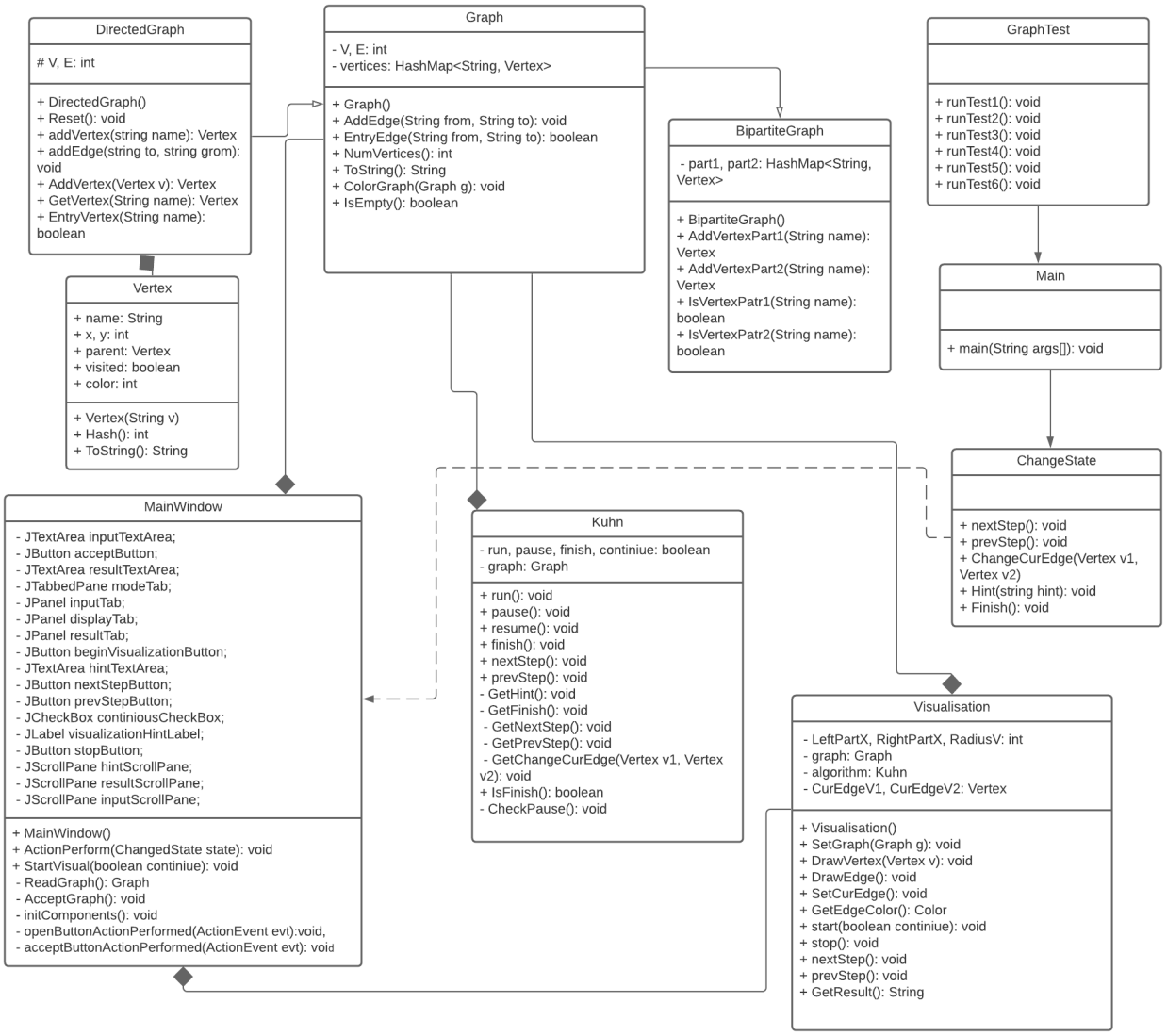

Рисунок 1: UML-диаграмма классов

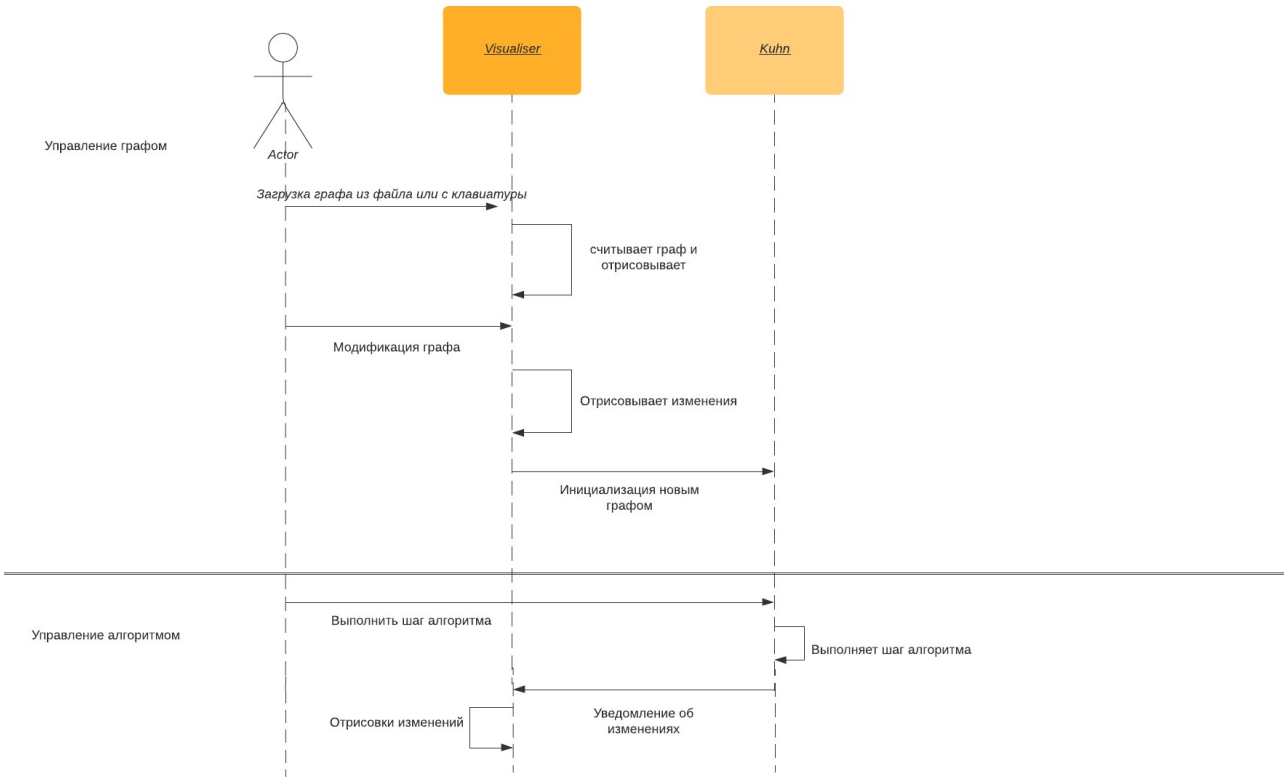

Рисунок 2: UML-диаграмма последовательностей

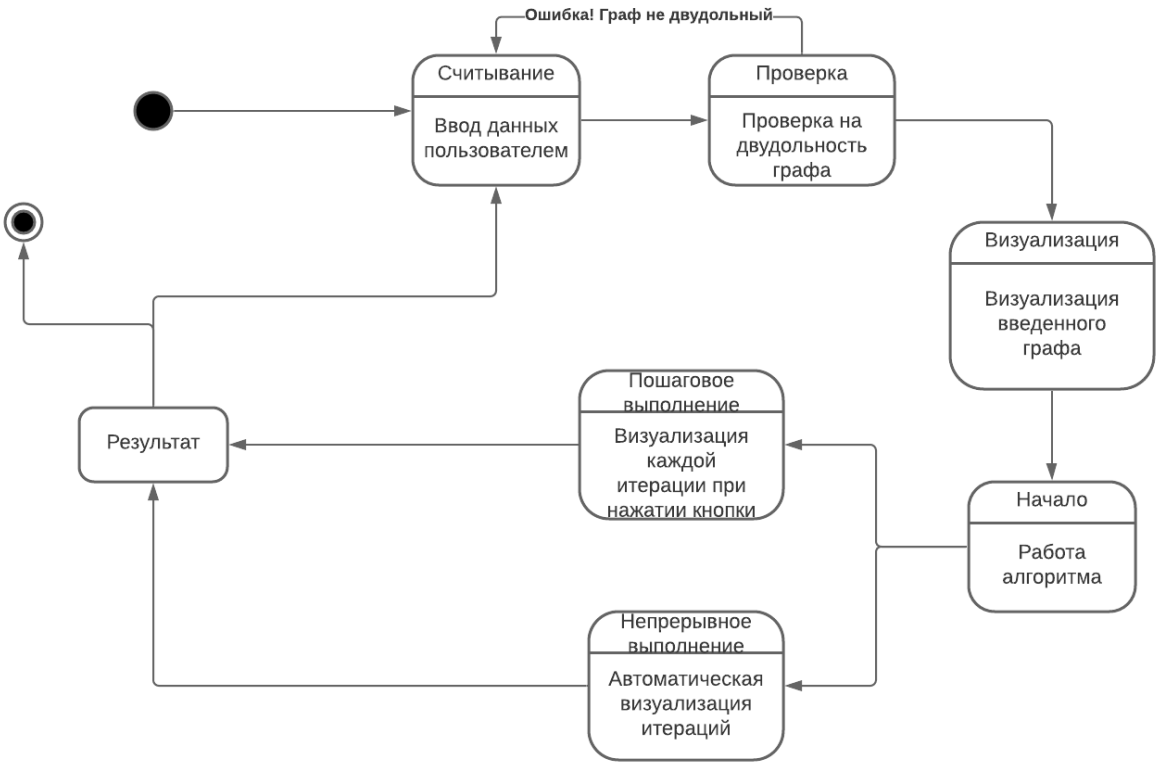

Рисунок 3: UML-диаграмма состояний

4. ПРОТОТИП GUI

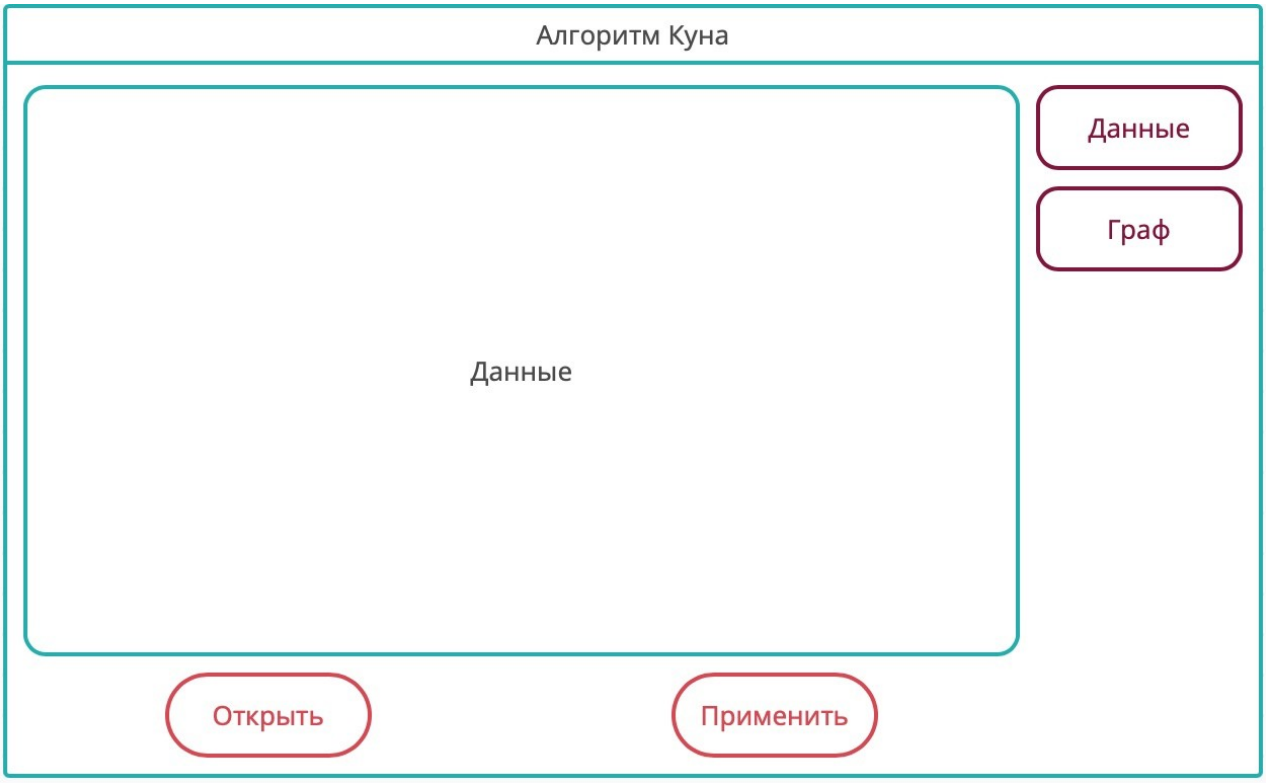

Рисунок 4: Прототип GUI

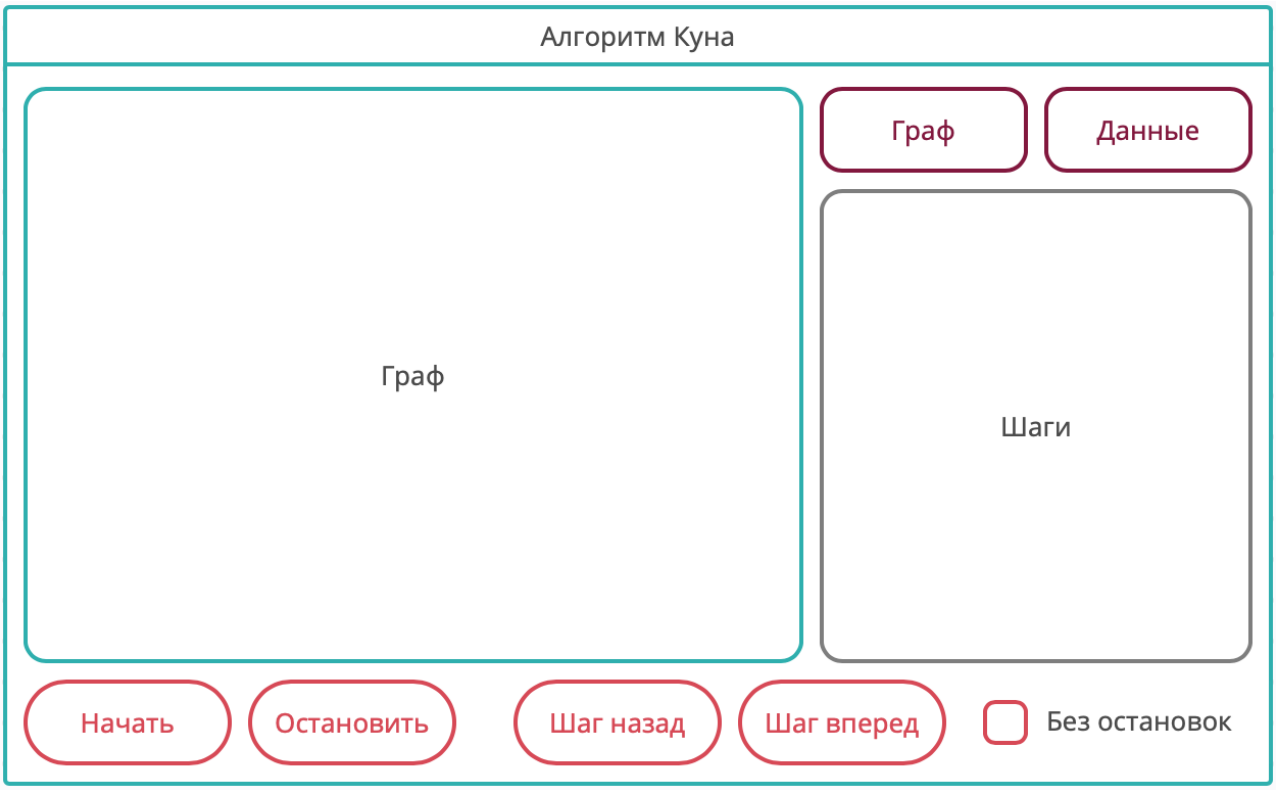

Рисунок 5: Прототип GUI

5. ОПИСАНИЕ АЛГОРИТМА

5.1 Действия программы

Программа выполняет следующую последовательность действий:

1) Считывание входных данных.

2) Проверка графа на двудольность.

- В каждой компоненте связности выбрать любую вершину и помечать оставшиеся вершины во время обхода графа в ширину поочередно, как четные и нечетные. Если при этом не возникает конфликта, то все нечетные вершины относятся к одной доле, а все четные — к другой.

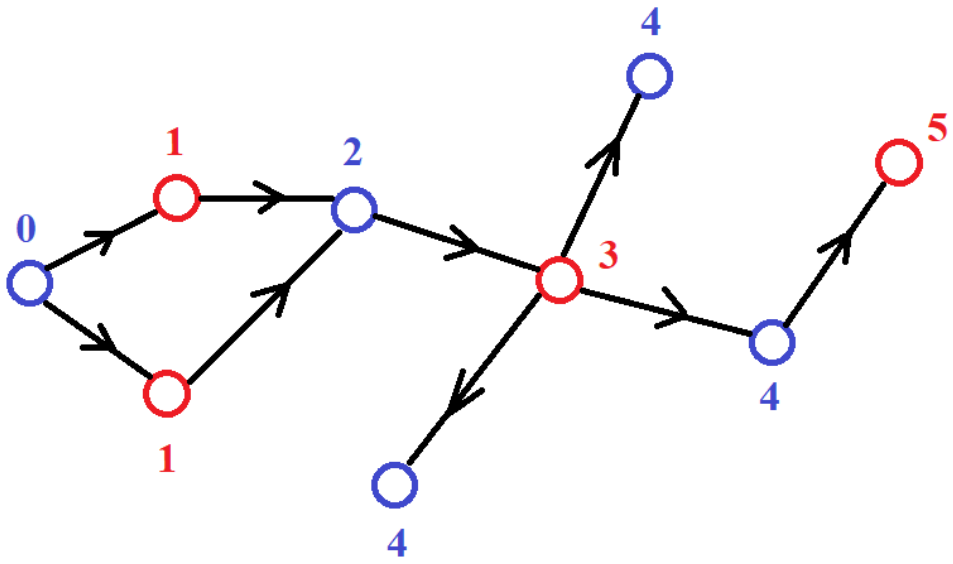

Рисунок 6: Проверка графа на двудольность

3) Если проверка пройдена, выполняется визуализация алгоритма, иначе — вывод сообщения об ошибке.

4) Вывод результатов работы алгоритма.

5.2 Поиск наибольшего паросочетания

Для поиска наибольшего паросочетания используется ***алгоритм Куна***.

Сначала возьмём пустое паросочетание, а потом — пока в графе удаётся найти увеличивающую цепь, — будем выполнять чередование паросочетания вдоль этой цепи, и повторять процесс поиска увеличивающей цепи. Как только такую цепь найти не удалось — процесс останавливаем, — текущее паросочетание и есть максимальное.

Поиск увеличивающей цепи осуществляется с помощью специального обхода в глубину или ширину (обычно в целях простоты реализации используют именно обход в глубину). Изначально обход в глубину стоит в текущей ненасыщенной вершине *v* первой доли. Просматриваем все рёбра из этй вершины, пусть текущее ребро — это ребро *(v,to).* Если вершина *to* ещё не насыщена паросочетанием, то, значит, мы смогли найти увеличивающую цепь: она состоит из единственного ребра *(v,to)*; в таком случае просто включаем это ребро в паросочетание и прекращаем поиск увеличивающей цепи из вершины *v*. Иначе, — если *to* уже насыщена каким-то ребром *(p,to)* то попытаемся пройти вдоль этого ребра: тем самым мы попробуем найти увеличивающую цепь, проходящую через рёбра *(v,to), (to,p).* Для этого просто перейдём в нашем обходе в вершину *p* — теперь мы уже пробуем найти увеличивающую цепь из этой вершины.

Время работы

Алгоритм Куна можно представить как серию из` *n* запусков обхода в глубину/ширину на всём графе. Следовательно, всего этот алгоритм исполняется за время $O(n*m)$, что в худшем случае есть $O(n^3)$.

6. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

6.1 Использованные структуры данных

Граф задается списком ребер.

public class Vertex - создание вершины и ее раскраска при посещении.

public class DirectedGraph - класс, представляющий ориентированный невзвешанный граф вершин, со строковыми именами. Он поддерживает следующие операции: добавление ребра, добавление вершины, получение списка всех вершин, получение списка соседей вершины, проверка наличия вершины в графе, проверка наличия ребра в графе. Граф представлен списком ребер.

public class Graph - наследуется от DirectedGraph. Класс, представляющий неориентированный невзвешанный граф вершин, со строковыми именами.

public class BipartiteGraph - наследуется от Graph. Класс, представляющий неориентированный невзвешанный двудольный граф.

public class Kuhn - Реализация алгоритма Куна для поиска наибольшего паросочетания в графе.

public class MainWindow – класс, реализующий графический интерфейс (GUI).

public class GraphTest – класс для тестирования

public class Visualizer - визуализация алгоритма.

6.2 Основные методы

public class Vertex

public int hashCode() - возвращает хэш текущей вершины. Имя вершины уникально в графе, поэтому оно используется в качестве хеша. Overrides: hashCode in class java.lang.Object

public int compareTo(Vertex other) - метод, необходимый для сортировки вершин в HashMap. Сравнивает текущую вершину с вершиной other. Функция Возвращает 0, если текущая вершина совпадает с other, отрицательное число, если текущая вершина "меньше" other, положительное число, если текущая вершина "больше" other

public java.lang.String toString() - возвращает имя вершины

public class DirectedGraph

Public void reset() - обнуление состояния вершин графа

public Vertex addVertex(java.lang.String name) — добавление новой вершины без соседей по имени (если в графе нет вершины с таким же именем). Фукнция возвращает добавленную вершину, либо уже существующую в графе вершину

public Vertex addVertex(Vertex v)- добавление в граф вершину без соседей (если в графе нет вершины с таким же именем). Функуция возвращает добавленную вершину, либо уже существующую в графе вершину

public Vertex getVertex(java.lang.String name) - возвращает вершину по соответствующему имени

public boolean EntryVertex(java.lang.String name) - проверяет вхождение вершины в граф

public boolean EntryVertex(Vertex v) - проверяет вхождение вершины в граф

public void addEdge(java.lang.String from,java.lang.String to) - добавление ребра в граф. Не добавляет ребро, если такое же ребро уже существует. Если какой-либо вершины нет в графе, то добавляет ее.

public boolean EntryEdge(Vertex from, Vertex to) — проверка на наличие ребра

public boolean EntryEdge(java.lang.String from,java.lang.String to) — проверка на наличие ребра from-to в графе

public java.lang.Iterable<Vertex> getVertices() - возвращает итератор по всем вершинам графа

public java.lang.Iterable<Vertex> getNeighbours(Vertex v) - возвращает список соседей заданной вершины

public java.lang.Iterable<Vertex> getNeighbours(java.lang.String name) - возвращает список соседей вершины с заданным именем

public Vertex getUnvisitedVertex() - возвращает непосещенную вершину графа. Если все вершины были посещены, возвращает null

public int numVertices() - возвращает число вершин в графе

public int numEdges() - возвращает число ребер в графе

public java.lang.String toString() - возвращает строковое представление графа в виде списка смежности

public boolean isEmpty() - проверка на пустоту вершины

public class Graph

Класс Graph, представляющий неориентированный невзвешанный граф вершин со строковыми именами

public void addEdge(String from, String to) - добавляет ребро в граф. Не добавляет ребро, если такое же ребро уже существует. Если какой-либо вершины нет в графе, добавляет ее

public class BipartiteGraph

Класс BipartiteGraph, представляющий неориентированный невзвешанный двудольный граф.

Public Vertex addVertexPart1(java.lang.String name) - добавляет новую вершину по имени в 1-ю долю графа. Вершина не добавляется в граф, если в графе уже есть вершина с таким же именем

public Vertex addVertexPart1(Vertex v) - добавляет вершину в 1-ю долю графа. Вершина не добавляется в граф, если в графе уже есть вершина с таким же именем

public Vertex addVertexPart2(java.lang.String name) - добавляет новую вершину по имени во 2-ю долю графа. Вершина не добавляется в граф, если в графе уже есть вершина с таким же именем

public Vertex addVertexPart2(Vertex v) - добавляет вершину во 2-ю долю графа. Вершина не добавляется в граф, если в графе уже есть вершина с таким же именем

public Vertex addVertexOppositePart(java.lang.String addedVertex, java.lang.String opposedVertex) - добавляет вершину addedVertex в долю, противоположную opposedVertex

public Vertex addVertex(Vertex v) - добавляет вершину в 1-ю долю графа Вершина не добавляется в граф, если в графе уже есть вершина с таким же именем. Overrides: addVertex in class DirectedGraph

public boolean isVertexPart1(java.lang.String name) - проверяет вхождение вершины в 1-ю долю

public boolean isVertexPart2(java.lang.String name) - проверяет вхождение вершины во 2-ю долю

public boolean isVertexPart1(Vertex v) - проверяет вхождение вершины в 1-ю долю

public boolean isVertexPart2(Vertex v) - проверяет вхождение вершины во 2-ю долю

public int getVertexPart(java.lang.String name) - возвращает номер доли, которой принадлежит вершина. Если вершина не принадлежит ни одной доле, возвращает 0

public java.lang.Iterable<Vertex> getPart1Vertices() - возвращает итератор на вершины 1-й доли

public java.lang.Iterable<Vertex> getPart2Vertices() - возвращает итератор на вершины 2-й доли

public void addEdge(java.lang.String from, java.lang.String to) - добавляет ребро в граф. Если вершины находятся в одной доле, бросает исключение. Не добавляет ребро, если такое же ребро уже существует. Если какой-либо вершины нет в графе, добавляет ее. Overrides: addEdge in class Graph

Public java.lang.String toString() - возвращает строковое представление графа в виде списка смежности

public class Kuhn

Реализация алгоритма Куна для поиска наибольшего паросочетания в графе.

Public void run() - запуск алгоритма

public java.lang.String getMatching() - текстовое представление текущего паросочетания

public boolean isFinished() - возвращает true, если алгоритм завершил работу, иначе false

public void pause() - выставляет флаг приостановки алгоритма

public void resume() - продолжает выполнение алгоритма

public void stop() - выставляет флаг завершения алгоритма

public void step() - выполняет один шаг алгоритма

private void sendMatchingChanged() - смена паросочетания

private void sendActiveEdgeChanged(Vertex v1, Vertex v2) — смена активного ребра

private void sendHint(String hint, int depth) - вывести подсказу

public void addHintStep(String hint) — сохранение подсказки на очередном шаге алгоритма

public void addGraphStep(BipartiteGraph biGraph) — сохранение графа на очередном шаге

public class MainWindow

Был частично реализован графический интерфейс. Он создавался при помощи конструктора JForm. Для этого были добавлены следующие элементы:

private JPanel mainPanel – основа интерфейса.

private JTabbedPane modeTab – контейнер, реализующий перемещение по вкладкам программы.

private JPanel inputTab, private JPanel displayTab, private JPanel resultTab – панели, представляющие собой вкладки программы – ввод данных, визуализация и результат соответственно.

На вкладке «Ввод данных» присутствует текстовое поле - p*rivate JTextArea inputTextArea*, нужное для считывания списка ребер графа. Также на данной вкладке находятся 2 кнопки: *private JButton openButton* и *private JButton acceptButton*.

Первая кнопка реализует чтение информации о графе из файла. При нажатии открывается диалоговое окно с выбором входного файла. В реализации данной кнопки обрабатываются 2 исключения: FileNotFoundException(файл не найден) и IOException(исключение ввода-вывода). Если файл корректно открылся, то данные из него вписываются в inputTextArea. Вторая кнопка принимает данные из inputTextArea и выбрасывает исключение, если inputTextArea пуста. Данные никуда не передаются, так как еще не реализовано взаимодействие алгоритма и графического интерфейса.

На вкладке «Визуализация» находятся 4 кнопки: *private JButton beginVisualizationButton* - для начала визуализации алгоритма, *private JButton nextStepButton* - для следующего шага алгоритма, *private JButton prevStepButton* – для предыдущего шага алгоритма, *private JButton stopButton* – для остановки алгоритма. Данные кнопки отключены, пока пользователь не введет данные графа. Также на вкладке находятся *private JCheckBox continiousCheckBox* – для пошагового выполнения алгоритма, *private JTextArea hintTextArea* – для пояснения шагов алгоритма.

На вкладке «Результат» находится private *JTextArea resultTextArea* – для вывода результата работы алгоритма. Вышеперечисленные объекты не имеют функционала, из-за отсутствия взаимодействия графического интерфейса с алгоритмом.

public class GraphTest

Был реализован класс *GraphTest,* в котором содержатся различные тесты для проверки корректной работы программы.

В первом, втором и третьем тестах проверяется работоспособность реализованного алгоритма Куна. Для этого вручную заполняется граф g, после чего над ним выполняется алгоритм поиска максимального паросочетания. Для

сравнения полученного результата с ожидаемым используется *Assertions.assertEquals()*. Как параметры в него передаются значения, которые необходимо сопоставить. Утверждается, что ожидаемое и действительное равны. Если оба значения равны нулю, они считаются равными.

В четвертом, пятом и шестом тестах проверяется правильность выбрасывания исключений. Тестирование производится с помощью конструкции *try-catch*. Если исключение возникает до блока *try* – тест падает, так получается информация, что в коде возникли проблемы. Тест успешно завершается только тогда, когда тестируемая функция выбрасывает исключение нужного типа.

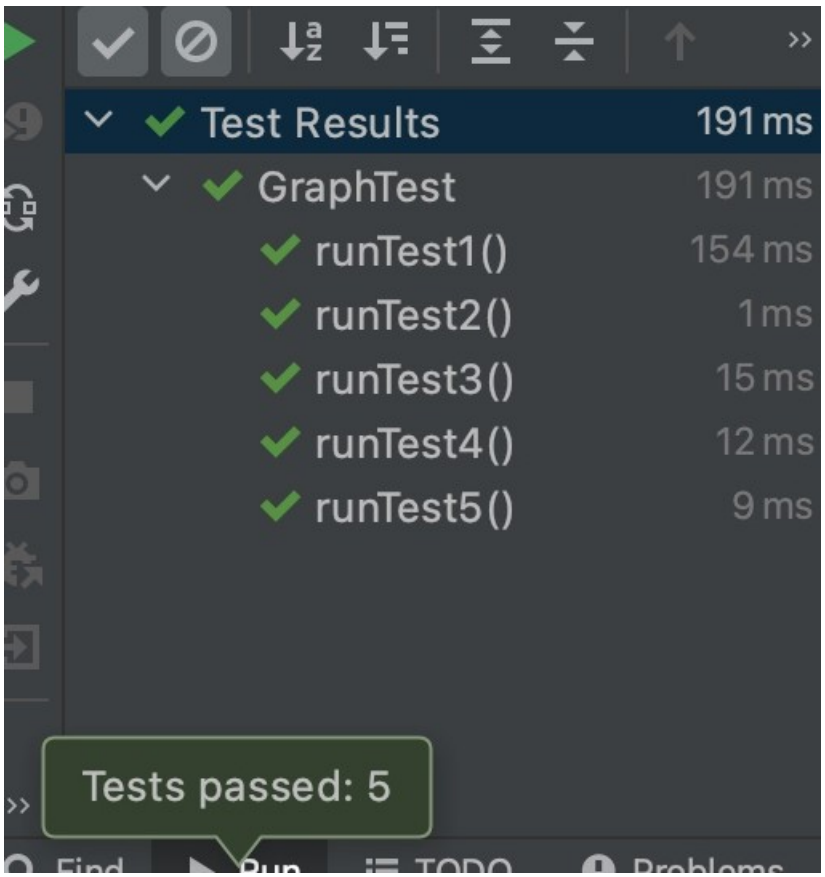

Рисунок 7: Успешное тестирование