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 Java

Студентка гр. 8382 _____ Наконечная А.Ю.

Студент гр. 8382 _____ Никитин А.Е.

Студентка гр. 8381 _____ Бердникова А.А.

Руководитель _____ Размочаева Н.В.

Санкт-Петербург
2020

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студентка Наконечная А.Ю. группы 8382

Студент Никитин А.Е. группы 8382

Студентка Бердникова А.А. группы 8381

Тема практики: визуализация алгоритмов на языке Java

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.

Тема проекта: Мосты графа.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 12.07.2020

Дата защиты отчета: 12.07.2020

Студентка	_____	Наконечная А.Ю.
Студент	_____	Никитин А.Е.
Студентка	_____	Бердникова А.А.
Руководитель	_____	Размочаева Н.В.

АННОТАЦИЯ

В ходе выполнения задания учебной практики была реализована программа, предназначенная для нахождения и удаления мостов графа. Поиск мостов графа осуществляется с использованием алгоритма поиска в глубину.

Разработанная программа детально показывает этапы работы алгоритма при нахождении мотов графа. Программа была написана на языке программирования Java, в среде разработки IntelliJ Idea.

SUMMARY

During the execution of the task of educational practice has been implemented a program designed to locate and remove the bridges of the graph. The search for graph bridges is performed using the depth search algorithm.

The developed program shows in detail the steps of the algorithm when finding the graph motes. The program was written in the Java programming language, in the IntelliJ Idea development environment.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

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

Цели выполнения учебной практики:

1. Освоение среды программирования Java, особенностей и синтаксиса данного языка.
2. Изучение средств для реализации графического интерфейса
3. Получение таких навыков как:
   - разработка программ на языке Java;
   - работа с системой контроля версий и репозиториями;
   - командная работа.

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Формулировка задания

Разработать программу, которая визуализирует алгоритм нахождения мостов графа на языке программирования Java.

1.2. Формальное определение и сложность алгоритма

Пусть дан неориентированный граф. Мостом называется такое ребро, удаление которого делает граф несвязным (или, точнее, увеличивает число компонент связности). Требуется найти все мосты в заданном графе.

Неформально эта задача ставится следующим образом: требуется найти на заданной карте дорог все "важные" дороги, т.е. такие дороги, что удаление любой из них приведёт к исчезновению пути между какой-то парой городов.

Ниже мы опишем алгоритм, основанный на поиске в глубину, и работающий за время O(n+m), где n — количество вершин, m — рёбер в графе.

1.3. Реализуемый алгоритм

Запустим обход в глубину из произвольной вершины графа; обозначим её через \rm root. Заметим следующий факт (который несложно доказать):

Пусть мы находимся в обходе в глубину, просматривая сейчас все рёбра из вершины v. Тогда, если текущее ребро (v,to) таково, что из вершины to и из любого её потомка в дереве обхода в глубину нет обратного ребра в вершину v или какого-либо её предка, то это ребро является мостом. В противном случае оно мостом не является. (В самом деле, мы этим условием проверяем, нет ли другого пути из v в to, кроме как спуск по ребру (v,to) дерева обхода в глубину.)

Теперь осталось научиться проверять этот факт для каждой вершины эффективно. Для этого воспользуемся "временами входа в вершину", вычисляемыми алгоритмом поиска в глубину.

Итак, пусть tin[v] — это время захода поиска в глубину в вершину v. Теперь введём массив fup[v], который и позволит нам отвечать на вышеописанные запросы. Время fup[v] равно минимуму из времени захода в саму вершину tin[v], времён захода в каждую вершину p, являющуюся концом

некоторого обратного ребра (v,p), а также из всех значений fup[to] для каждой вершины to, являющейся непосредственным сыном v в дереве поиска:

$$fup[v] = \min \begin{cases} tin[v], & \\ tin[p], & \text{for all (v,p) — back edge} \\ fup[to], & \text{for all (v,to) — tree edge} \end{cases}$$

(здесь "back edge" — обратное ребро, "tree edge" — ребро дерева)

Тогда, из вершины v или её потомка есть обратное ребро в её предка тогда и только тогда, когда найдётся такой сын to, что fup[to] \le tin[v]. (Если fup[to] = tin[v], то это означает, что найдётся обратное ребро, приходящее точно в v; если же fup[to] < tin[v], то это означает наличие обратного ребра в какого-либо предка вершины v.)

Таким образом, если для текущего ребра (v,to) (принадлежащего дереву поиска) выполняется fup[to] > tin[v], то это ребро является мостом; в противном случае оно мостом не является.

1.4. Псевдокод

Обозначения:

enter[v] - время захода поиска в глубину в определённую вершину v

ret[v] - минимум из: времени захода в саму вершину enter[v], времён захода в каждую вершину p, являющуюся концом некоторого обратного ребра (v,p), а также из всех значений ret[to] для каждой вершины to, являющейся непосредственным сыном v

comp - список вершин в текущей компоненте связности

bridges - список, содержащий мосты

```
function dfsBridges(v):
  time = time + 1
  used[v] = true
  enter[v] = time
  ret[v] = time
  for всех u смежных с v
    if (v, u) — обратное ребро
      ret[v] = min(ret[v], enter[u])
    if !used[u]
      dfsBridges(u)
      ret[v] = min(ret[v], ret[u])
      if ret[u] > enter[v]
        ребро (v, u) — мост
            addBridge(v, u, bridges)
function findBridges():
  time = 0
  for для всех вершин
    used[i] = false
  for для всех вершин
    if !used[i]
      dfsBridges(i)
function dfsComps(v):
  used[v] = true
  comp.push_back(v)
  for всех u смежных с v
```

```
    if !used[u]
      dfsComps(u);
fuction findComps():
  for для всех вершин
    used[i] = false
  for для всех вершин
    if !used[i]
      clear(comp)
      dfsComps(i)
function deleteBridges():
  findBridges()
  for каждого моста с вершинами v, u из bridges
    deleteEdge(v, u)
```

2. СПЕЦИФИКАЦИЯ

2.1. Требования к интерфейсу пользователя

Пользовательский интерфейс должен представлять собой диалоговое окно, содержащее набор кнопок, предназначенных для управления состоянием программы. Набросок GUI представлен на рис.1.

Основное диалоговое окно должно состоять из:

- Рабочей области для построения графа.
- Кнопки «Удалить мосты», которая должна удалить все мосты графа.
- Кнопки «Удалить ребро», которая должна удалить выбранное пользователем ребро графа.
- Кнопки «Найти мосты», которая применяет алгоритм к графу из рабочей области
- Кнопки «Пошаговый прогон», которая активирует кнопки «Вперед» и «Назад».
- Кнопки «Вперед», которая должна отобразить следующую итерацию алгоритма.
- Кнопки «Назад», которая должна отобразить предыдущую итерацию алгоритма

- Кнопки «Сохранить...», которая должна позволять пользователю сохранить созданный в рабочей области граф.

Всплывающее первоначальное диалоговое окно:

- Кнопки «Информация», показывающей пользователю информацию о работе алгоритма и вводе данных.
- Окна с изменяющимся значением для определения пользователем количества вершин графа.
- Кнопки «Загрузить граф», которая должна позволять пользователю загрузить граф из файла.

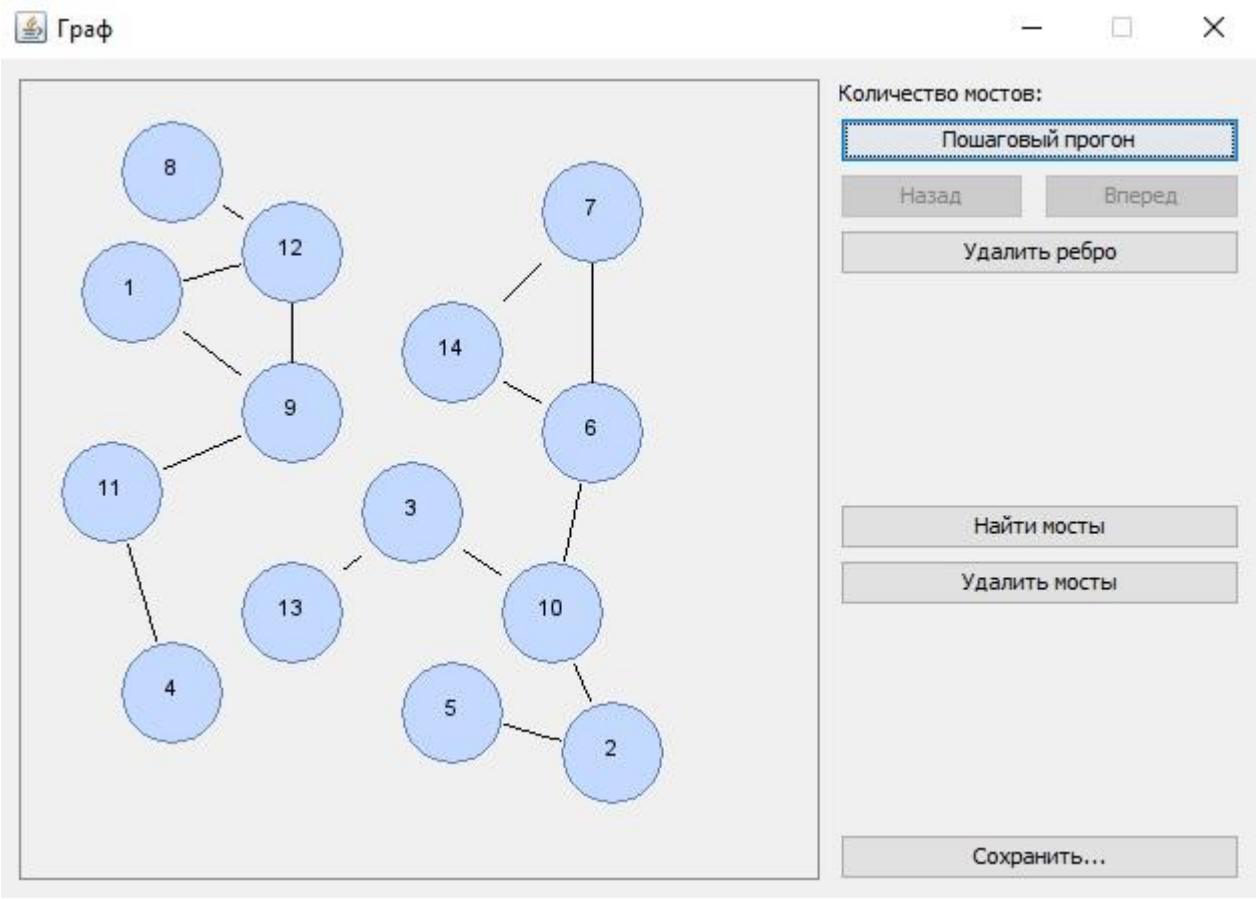

Рисунок 1- Графика программы

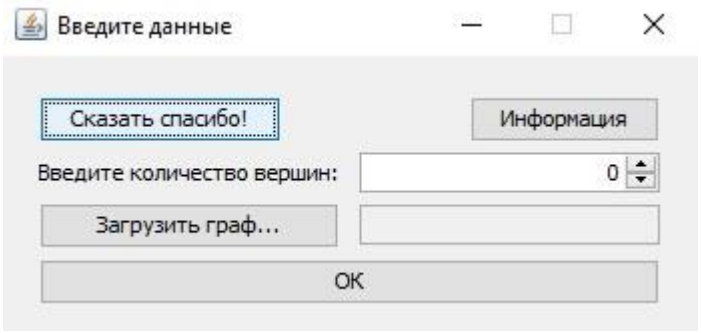

Рисунок 2 - Графика программы

2.2. Требования к вводу исходных данных

На вход алгоритму должен подаваться невзвешенный неориентированный граф. Именем вершины может быть всё угодно. Область создания графа предоставляет возможность ввести данные с файла или сгенерировать. При ручном вводе происходит взаимодействие с графическим представлением графа с помощью мыши. Изначально, появляются все вершины в одной точке, с помощью зажатия и переноса в другое место программы мы

сможем увидеть все вершины. Далее кликом по вершине и последующем зажатии и соединения первой вершины с любой другой создается ребро, а через контекстное меню вершины можно будет удалить его. При считывании с файла надо нажать соответствующую кнопку в самом начале программы.

3. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

3.1. План разработки

План разработки программы разбит на следующие шаги:

1-ая неделя:

1 июля (среда) / 2 июля (четверг) - сдача спецификации, распределения ролей и плана разработки

3 июля (пятница) / 5 июля (воскресенье) - создание прототипа
(Функции алгоритма: добавление вершины, наличие вершины, наличие ребра, добавление ребра, удаление ребра, поиск мостов, удаление мостов); (Графика: Окно создания графа, окно графа);

2-я неделя:

6 июля (понедельник) - сдача прототипа

7 июля (вторник) - создание 1-ой версии проекта
(Функции алгоритма: создание функций "шаг вперёд", "шаг назад");
(Графика: изучение основных функций, нужных для представления графа благодаря JGraph);

8 июля (среда) - сдача 1-ой версии проекта

9 июля (четверг) - создание 2-ой версии проекта (Функции алгоритма: поиск компонент связности в неориентированном графе); (Графика: реализация представления графа, реализация добавления/удаления рёбер, итоговая корректировка кнопок, unit-тестирование);

10 июля (пятница) - сдача 2-ой версии проекта и отчёта

11 июля (суббота) / 13 июля (понедельник) - создание 3-ей версии проекта с последующей сдачей (в случае необходимости)

3.2. Распределение ролей в бригаде

Роли в бригаде распределены следующим образом:

Наконечная Анастасия: алгоритмист;

Никитин Александр: представитель, фронтенд;

Бердникова Анастасия: тестировщик, документация;

1. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРТИМА

1.1 Структуры данных

Для создания графа были реализованы следующие классы: Graph и Edge.

А также граф хранится в структуре HashMap для выполнения некоторых функций. В классе BasicFunctions реализованы следующие методы:

- public void dfs() — обход рёбер с помощью поиска в глубину;
- public void findDfs() – инициализация структур для поиска в глубину;
- public void deleteEdge() – удаление рёбер в представлении графа (коллекция HashMap vertexMap);
- public boolean hasEdge() – функция возвращает true в том случае, когда есть ребро.
- public void printGraph() - метод печатает граф в представлении структуры HashMap;
- public void stepGo() – позволяет произвести пошаговое движение по графу;
- public void findBridgesButton() – осуществление поиска мостов после нажатия на кнопку «Поиск мостов»;
- public void deleteBridgesButton() - осуществление удаления мостов после нажатия на кнопку «Удаление мостов»;
- public void deleteEdgeButton() - осуществление удаления ребра после нажатия на кнопку «Удаление ребра»;
- private void deleteBridges() - осуществляет удаление мостов в структуре HashMap.
- private void findBridges() - осуществляет поиск мостов в структуре HashMap;

- public int getBridgesNum() - возвращает количество мостов.

1.2 Основные методы

Основные методы, необходимые для эффективной работы алгоритма Поиска Мостов, были написаны в классе BasicFunctions.

Поиск мостов происходит с помощью поиска в глубину. Как ранее было написано, если текущее ребро (v,to) таково, что из вершины to и из любого её потомка в дереве обхода в глубину нет обратного ребра в вершину v или какого-либо её предка, то это ребро является мостом. В противном случае оно мостом не является. С помощью нескольких коллекций HashMap получилось создать алгоритм, способный отследить все мосты в графе, а затем удалить их. Удаление ребра негативно не влияет на работу основных функций, что позволяет находить все необходимые компоненты графа.

2. ТЕСТИРОВАНИЕ

2.1 Мануальное тестирование

При запуске пробной версии программы всплывает диалоговое окно, представленное на рис.2

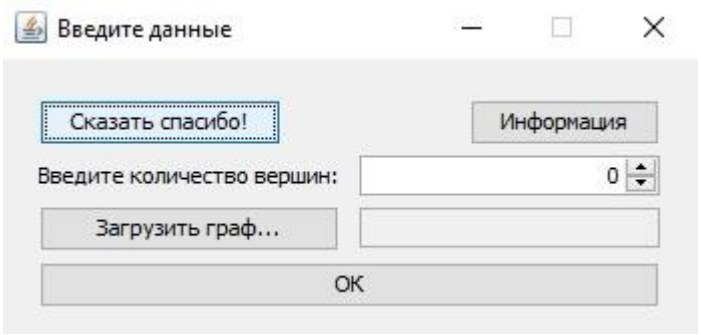

Рисунок 3 – Начальный экран программы

На этом этапе предстоит выбрать: загружать граф или же создавать самому. Для создания нужно выбрать желаемое количество вершин. Для загрузки нажать на соответствующую кнопку.

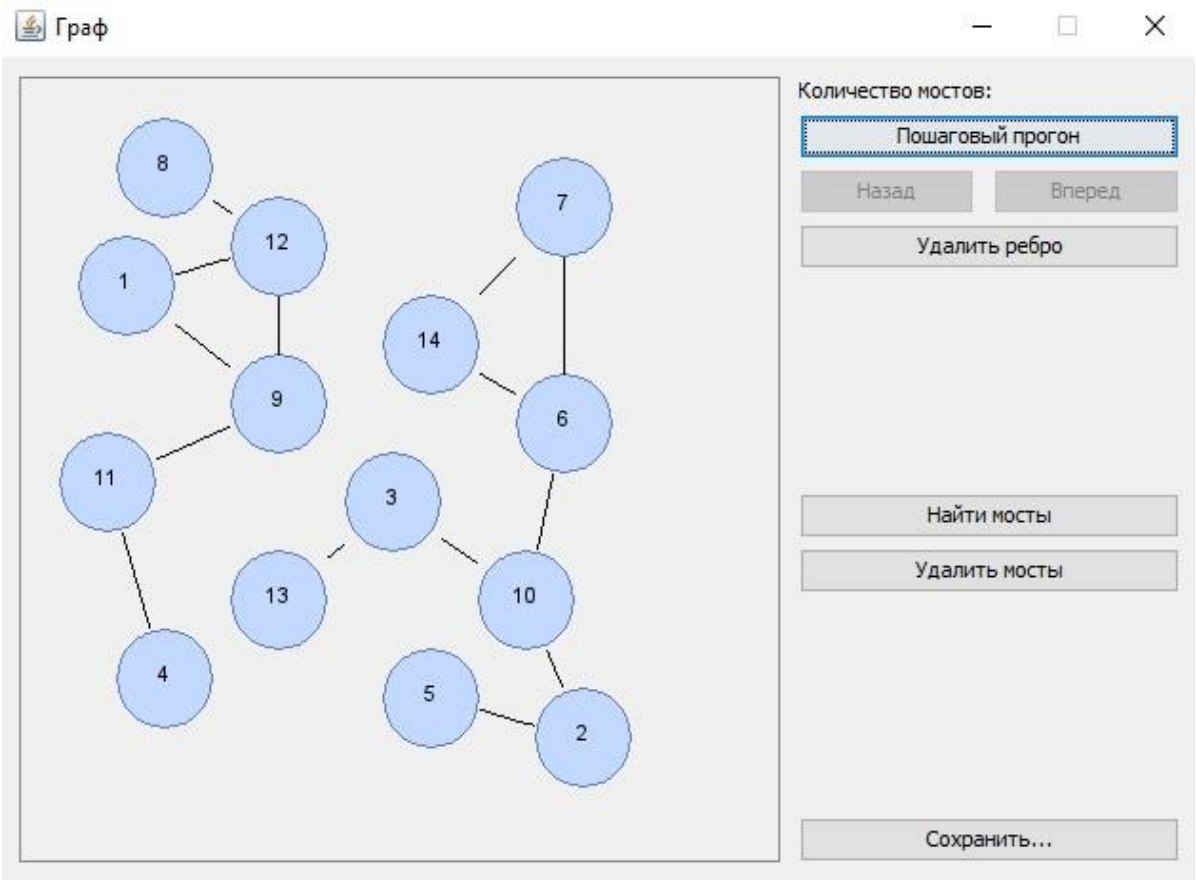

Рисунок 4 – Вид программы при загруженном граф

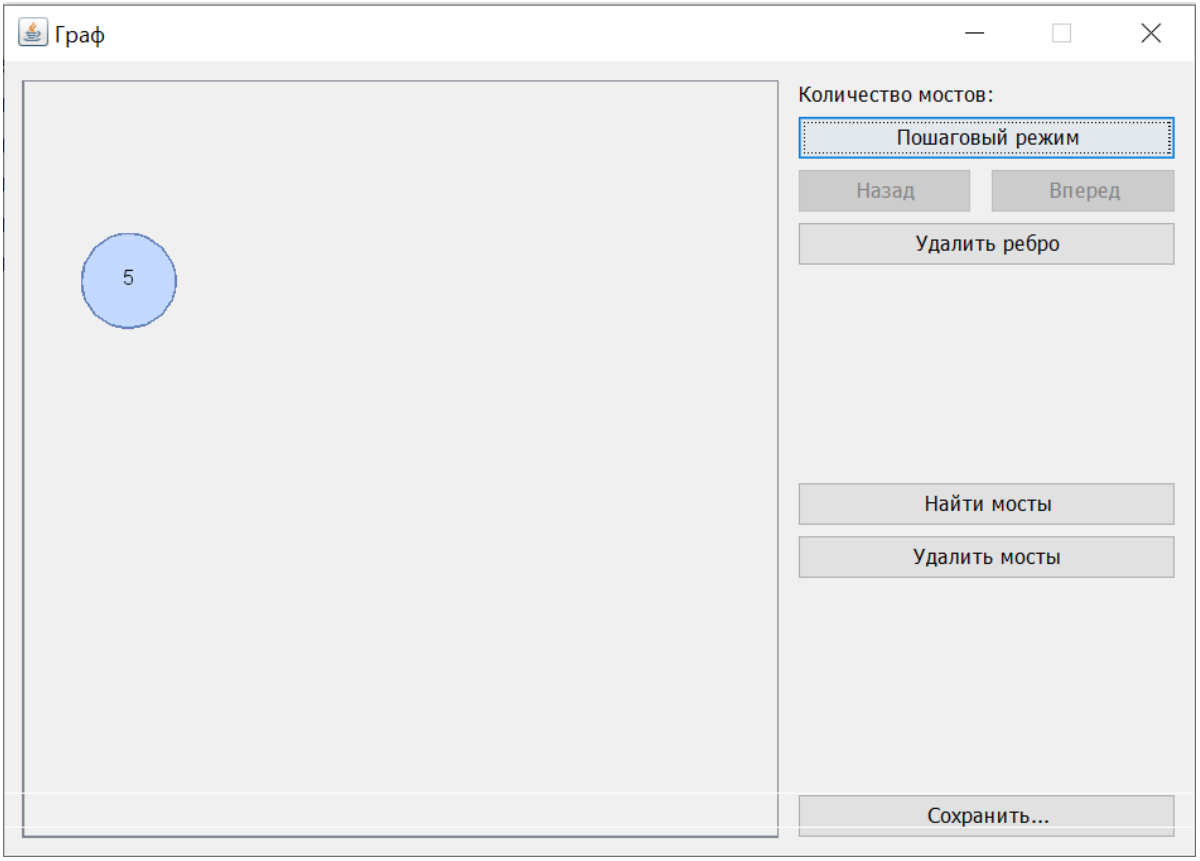

Рисунок 5 – Начальный вид программы при самостоятельном создании графа с 5 вершинами

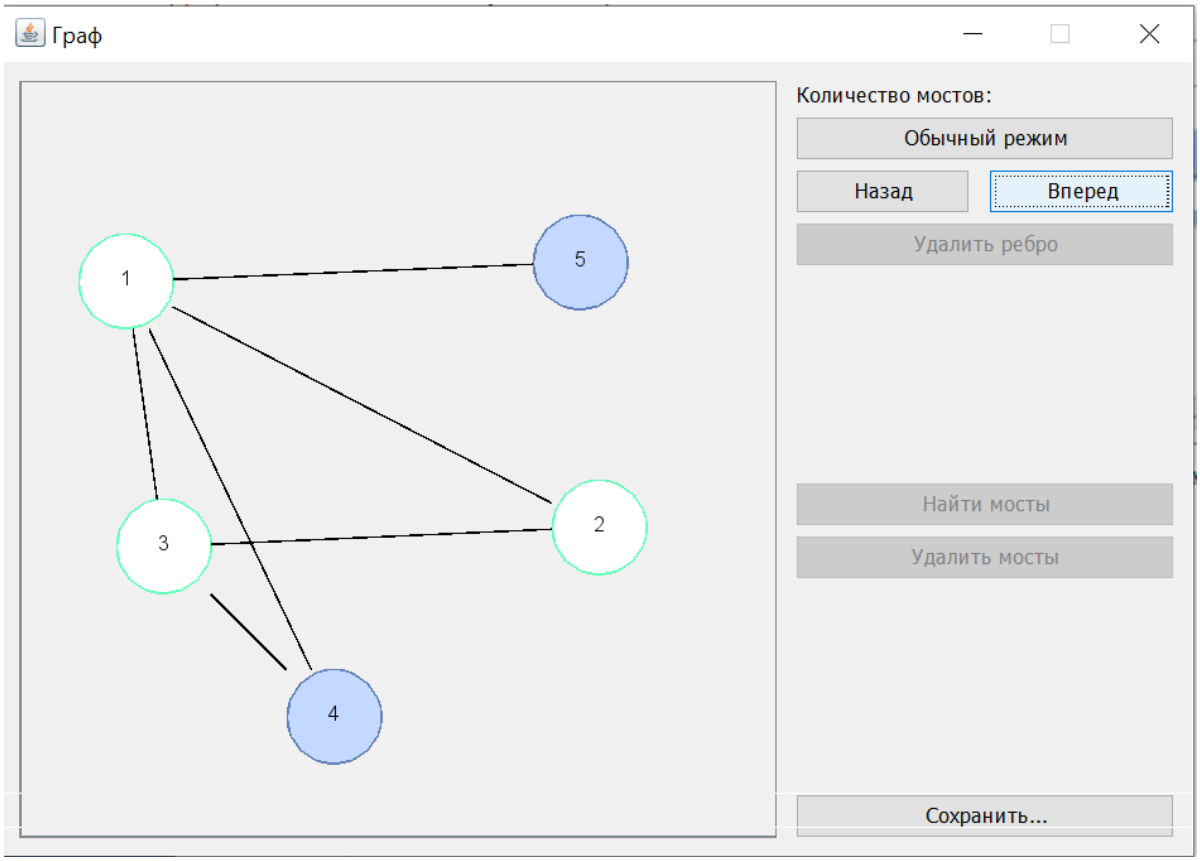

Рисунок 6 – Пошаговая демонстрация алгоритма

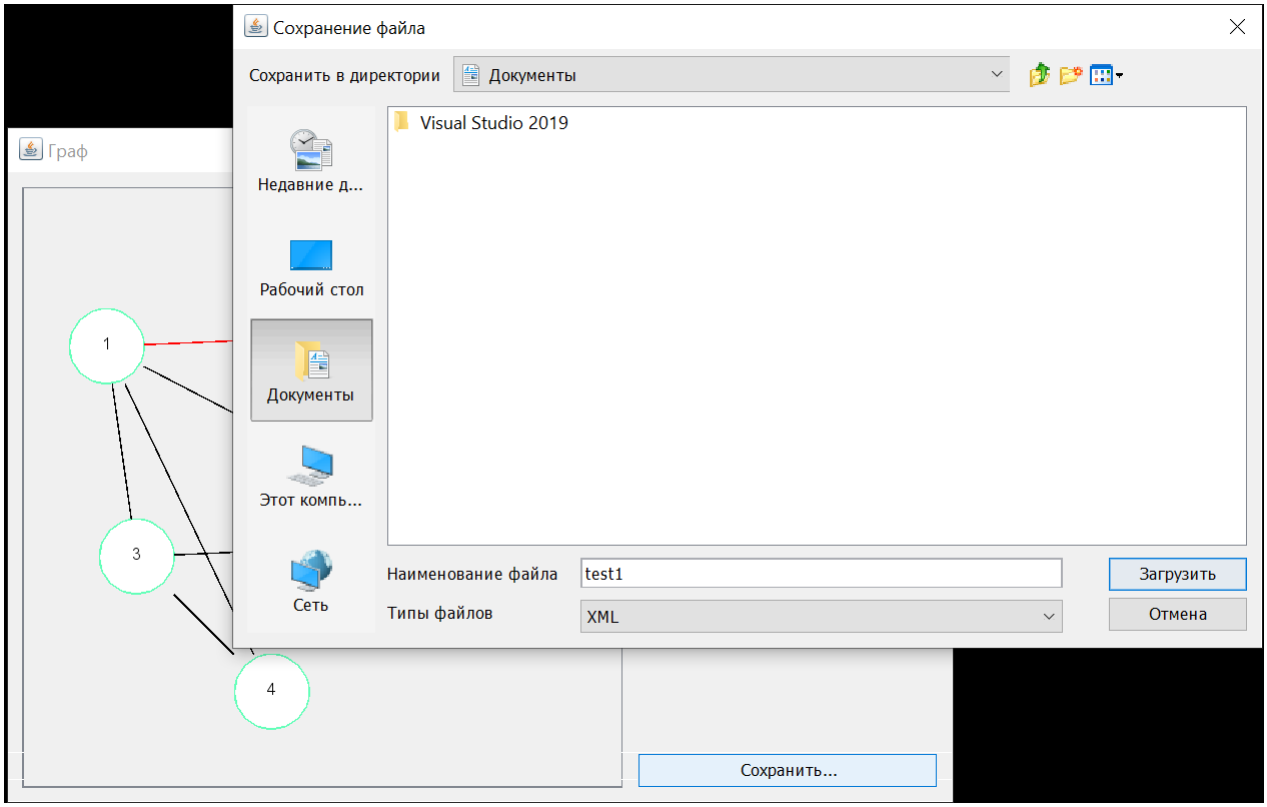

Рисунок 7 – Сохранение графа

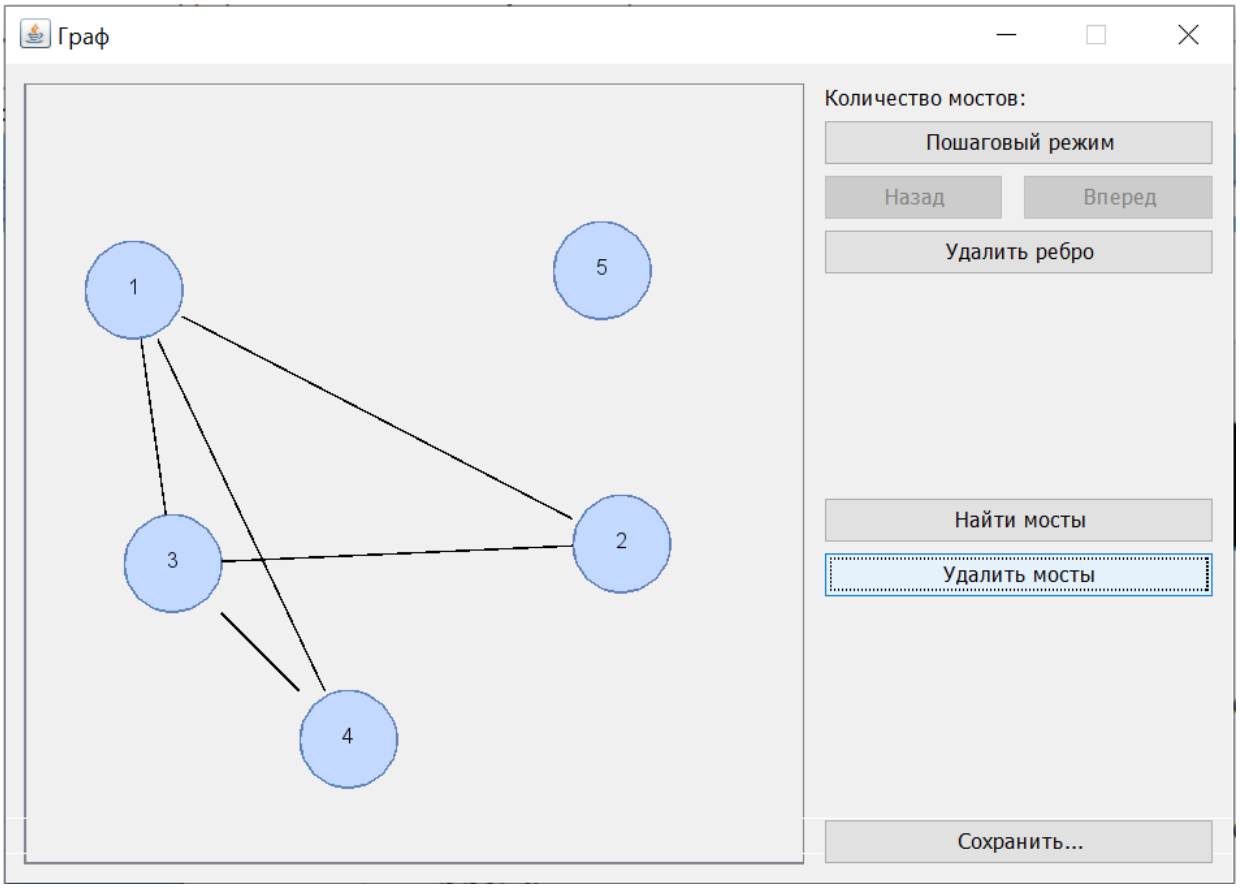

Рисунок 8 - Результат при выборе функции удаление мостов

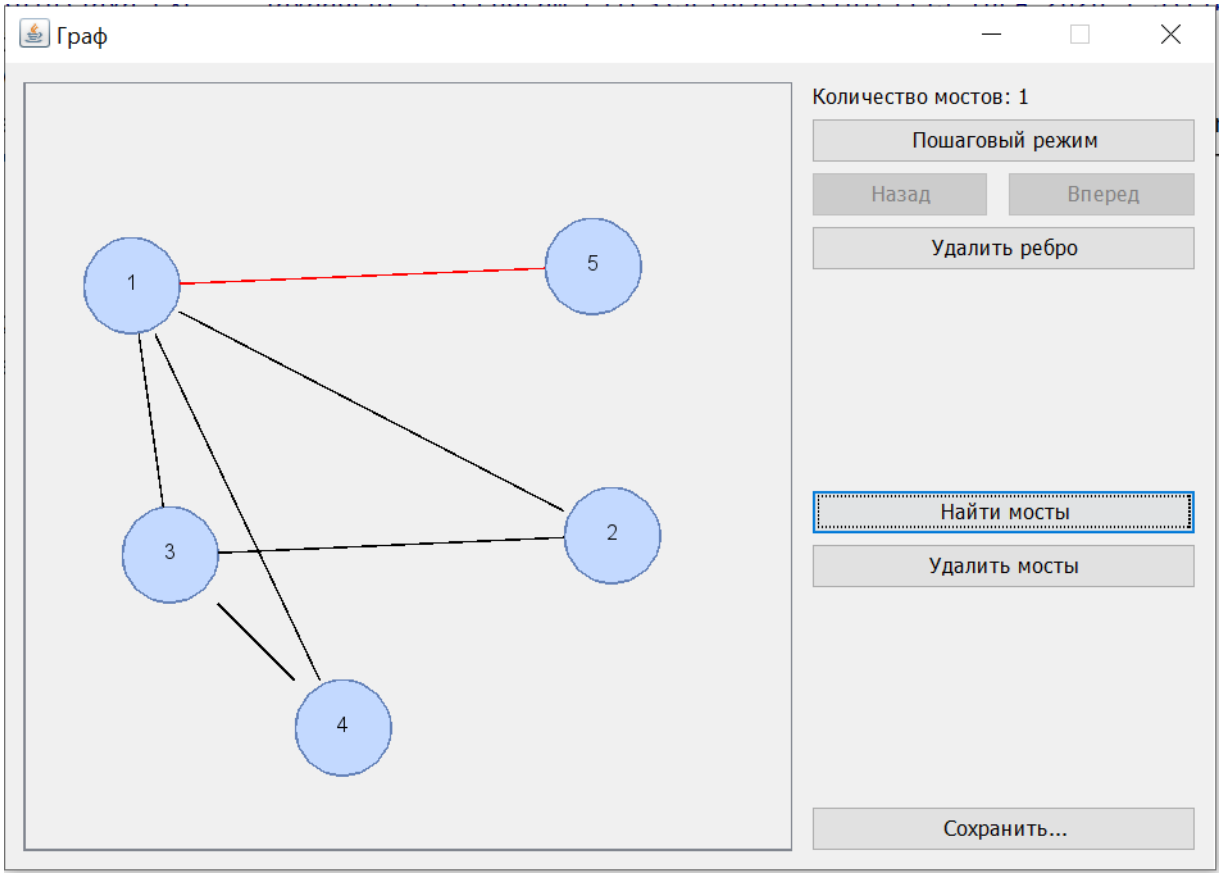

Рисунок 9 - Результат при выборе функции нахождения мостов

2.2 Unit тестирование программы

Для проведение автоматического тестирования программы использовалась библиотека junit 4. UNIT тесты были написаны с целью проверить основные моменты кода, а именно проверить корректность нахождения мостов, добавления ребра, удаления ребер и мостов.

После запуска Unit тестирования корректные тесты для него будут выглядеть так, как на рис.21

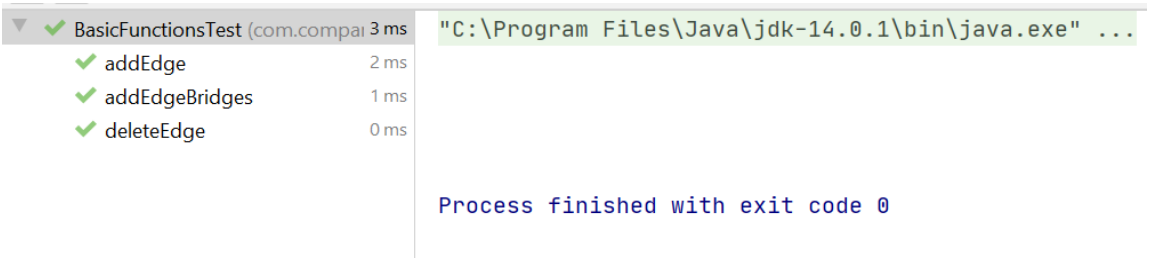

Рисунок 10 – Проверка тестов

UML-ДИАГРАММА

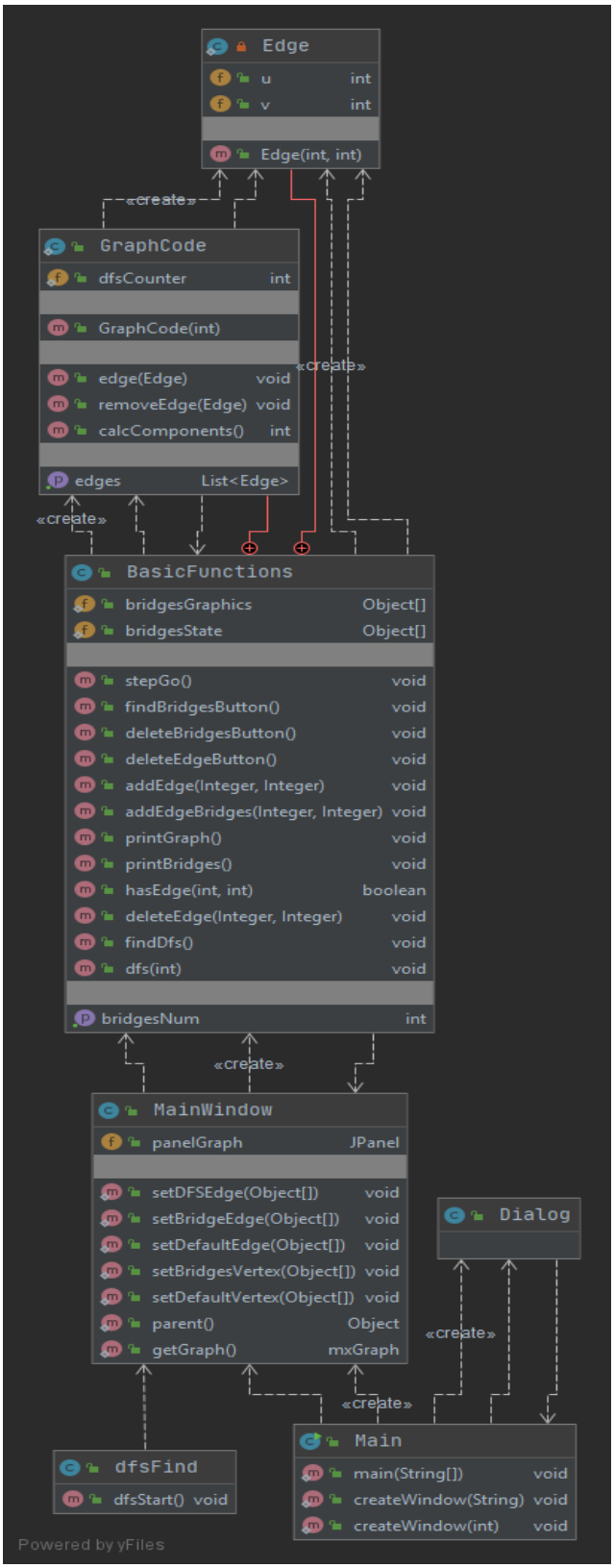

Рисунок 11 – UML-Диаграмма

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы мы ознакомились со средой разработки IntelliJ IDEA, а также языком программирования Java. Нами был успешно реализован алгоритм нахождения минимального остовного дерева, а также его визуализация и интерфейс самой программы. Все поставленные задачи были выполнены полностью и в срок. Также к основным требованиям нами были добавлены логирование, вывод/ввод из файла, изменение языка программы, изменение параметров визуализации, а также информация о создателях и самой программе.

Исходя из всего вышеперечисленного, можно сделать вывод, что поставленные задачи были выполнены успешно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Учебный курс по основам Java на Stepik: https://stepik.org/course/187/
2. Официальная документация к Java: https://docs.oracle.com/en/java/javase/
3. Java 8. Руководство для начинающих. Герберт Шилдт