ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Доцент департамента больших  данных и информационного поиска  ФКН НИУ ВШЭ,  канд. физ.-мат. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Л. Чернышев  « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. | УТВЕРЖДАЮ  Академический руководитель  образовательной программы  «Программная инженерия», профессор,  канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. |

|  |  |
| --- | --- |
| Инв. № подл. |  |
| Подп. и дата |  |
| Взам. Инв. № |  |
| Инв. № дубл. |  |
| Подп. и дата |  |

**Программа для нахождения**

**явного вида аналитических функций,**

**связанных с метрическими графами**

**Пояснительная записка**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.04.01-01 81 01-1-ЛУ**

Исполнитель

студент группы БПИ181

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/А.И. Уварова/

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

УТВЕРЖДЕН

RU.17701729.04.01-01 81 01-1 ЛУ

**Программа для нахождения**

|  |  |
| --- | --- |
| Инв. № подл. |  |
| Подп. и дата |  |
| Взам. Инв. № |  |
| Инв. № дубл. |  |
| Подп. и дата |  |

**явного вида аналитических функций,**

**связанных с метрическими графами**

**Пояснительная записка**

**RU.17701729.04.01-01 81 01-1**

**Листов 38**

**Содержание**

[**1.ВВЕДЕНИЕ** 3](#_Toc39538988)

[**1.1 Наименование программы** 3](#_Toc39538989)

[**1.2 Основания для разработки** 3](#_Toc39538990)

[**2.Назначение и область применения** 4](#_Toc39538991)

[**2.1 Назначение программы** 4](#_Toc39538992)

[**2.2 Краткая характеристика области применения** 4](#_Toc39538993)

[**3. Технические характеристики** 5](#_Toc39538994)

[**3.1 Постановка задачи на разработку программы** 5](#_Toc39538995)

[**3.2 Описание алгоритма и функционирования программы** 5](#_Toc39538996)

[**3.2.1 Описание алгоритма распределения амплитуд** 5](#_Toc39538997)

[**3.2.2 Описание реализации алгоритма распределения амплитуд** 5](#_Toc39538998)

[**3.2.3 Описание реализации алгоритма парсера** 9](#_Toc39538999)

[**3.4. Описание и обоснование выбора состава технических средств** 10](#_Toc39539000)

[**3.5** **Описание и обоснование выбора состава программных средств** 10](#_Toc39539001)

[**4.ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ** 11](#_Toc39539002)

[**5. ПРИЛОЖЕНИЯ** 12](#_Toc39539003)

[**ПРИЛОЖЕНИЕ 1** 12](#_Toc39539004)

[**ПРИЛОЖЕНИЕ 2** 13](#_Toc39539005)

[**ПРИЛОЖЕНИЕ 3.** 15](#_Toc39539006)

# **1.ВВЕДЕНИЕ**

## **1.1 Наименование программы**

Наименование программы – «Программа для нахождения явного вида аналитических функций,

связанных с метрическими графами».

Наименование на английском языке – «A program for finding the explicit form of analytical

functions associated with metric graphs».

## **1.2 Основания для разработки**

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 "Программная инженерия" и утвержденная академическим руководителем тема курсового проекта.

# **2.Назначение и область применения**

## **2.1 Назначение программы**

Программа предназначена для построения неориентированных конечных метрических графов и вычисления по ним аналогов дзета-функций Ихары, функции магнитуд, а также собственной функции на путях графа.

Программа главным образом предназначена для исследовательских целей. Дзета-функции Ихары играют большую роль в исследованиях в разных разделах математики (теории чисел, теория групп, теория графов, топология и т. д.). Магнитуда графа по сути является одной из его мер и несет в себе информацию, которую не содержат другие схожие с ней многочлены (многочлен Татта, графический матроид). Функция на путях графа — это самодельная функция, комбинирующая в себе информацию о всех возможных путях из конкретной вершины графа.

## **2.2 Краткая характеристика области применения**

Программа используется в качестве графического редактора графов, а также для вычисления выше перечисленных функций по построенным моделям.

Области применения: исследования в области теории графов, топологии, геометрии, теории чисел, использование в обучающих целях.

# **3. Технические характеристики**

## **3.1 Постановка задачи на разработку программы**

Задачи, которые выполнять программа:

1. Построение моделей конечных метрических неориентированных графов с кратными ребрами-дугами и LaTeX-весами.
2. Вычисление аналитических функций по построенным графам и вывод результата в LaTeX-формате в файл (.tex, .txt)
3. Сохранение построенных графов в файлы формата .graph.
4. Загрузка ранее сохраненных графов в программу.

## **3.2 Описание алгоритма и функционирования программы**

Далее будем обозначать конечный неориентированный граф как G, E — множество его ребер, V — множество вершин.

### **3.2.1 Описание реализации алгоритма вычисления функции магнитуд**

Обозначим минимальное расстояние между вершинами x, y графа G как d(x,y).

Положим что матрица (q — символьный аргумент) - это квадратная матрица , чьи ряды и столбцы определяются вершинами графа G и где (x, y)-й элемент равен

Формула для вычисления магнитуд по графу:

Алгоритм:

1) Вычисляются минимальные расстояния до всех вершин для каждой вершины графа. Для вычисления использован модифицированный под задачу алгоритм Дейкстры. В каждой вершине запоминаем текущее минимальное вещественное расстояние, а также храним сет символьных токенов, которые являются длинами ребер на минимальном пути. В конце сет токенов для каждой вершины преобразуется в символьное выражение, которое является суммой вещественной части, выраженной единственным числом, и иррациональных токенов в формате SymPy (изначально длины ребер в LaTeX-формате, затем с помощью парсера преобразуем в SymPy).

Алгоритм поиска минимальных расстояний для вершины:

private static void findMinDistance(Node n, String[] distances) {

PriorityQueue<Node> priorityQueue = new PriorityQueue<>(

Comparator.comparing(Node::getDijkstraDistance).reversed());

for (Node node : Graph.getInstance().getNodes()) {

if (node.equals(n))

node.setDijkstraDistance(0);

else

node.setDijkstraDistance(Double.MAX\_VALUE);

node.getDijkstraTexTokens().clear();

priorityQueue.add(node);

}

while (!priorityQueue.isEmpty()) {

Node minNode = priorityQueue.poll();

double distance = minNode.getDijkstraDistance();

distances[minNode.getNum() - 1] = n.equals(minNode) ? "0" : distance == Double.MAX\_VALUE ? "-1" :Parser.parseTexToSympy(minNode.getDijkstraTexTokens());

for (Map.Entry<Node, Pair<Double, String>> entry : minNode.getNeighboursAndDistances().entrySet()) {

double curVal = minNode.getDijkstraDistance() + entry.getValue().getKey();

Node node = entry.getKey();

if (curVal > 0 && node.getDijkstraDistance() > curVal) {

node.getDijkstraTexTokens().clear();

node.getDijkstraTexTokens().addAll(minNode.getDijkstraTexTokens());

node.getDijkstraTexTokens().add(entry.getValue().getValue());

node.setDijkstraDistance(curVal);

priorityQueue.remove(node);

priorityQueue.add(node);

}

}

}

}

2) Результаты записываются в матрицу минимальных расстояний и сохранятся в ресурсный файл формата .txt.

3) Из файла результаты вычитываются python-скриптом, который производит вычисление обратной матрицы по переданной и возвращает итоговый результат в LaTeX-формате.

### **3.2.2 Описание реализации алгоритма вычисления функции Ихары на вершинах**

Зададим ориентацию ребер графа (в программе согласно нумерации вершин) следующим образом: и получим 2|E| ориентированных ребер.

Пусть P — примитивный (, D - путь) путь в G.

где - ориентированное ребро,

Длина пути .

Класс эквивалентности [P] = {}

- ранг фундаментальной группы G,

Формула дзета-функции Ихары на вершинах:

, где u — комплексное число и |u| достаточно маленькая величина.

Для вычисления функции используются следующие формулы:

для невзвешенного графа,

для взвешенного графа.

Здесь I — единичная матрица, A — матрица смежности графа G, Q — диагональная матрица,

где j-е число на диагонали (степень j-й вершины — 1), W — матрица весов рёбер между вершинами.

Алгоритм:

1) По заданному графу вычисляются A/W, Q и . Каждая вершина хранит список своих соседей программно, поэтому матрицы просто находятся в цикле по каждой вершине. Длины ребер для W как и в предыдущем алгоритме записываются строками с иррациональной частью.

2) Вычисленные данные записываются в json-файл.

3) Python- скрипт записывает промежуточные матрицы и результат в строку формата .tex

4) Ответ записывается в выбранный пользователем файл формата .tex или .txt.

### **3.2.3 Описание реализации алгоритма вычисления функции Ихары на ребрах**

### Нормой ребер для пути P будем называть , где — достаточно маленькое комплексное число.

### Зададим матрицу рёбер для ориентированных рёбер из пункта 3.2.2. Она будет иметь размер , элемент (i, j) равен если ребро входит в ребро и , и 0 иначе.

### Формула функции Ихары на ребрах имеет вид:

### Формула для вычисления функции в программе:

### Алгоритм:

### 1) Вычисляем по заданному графу матрицу рёбер. Для этого ориентируем ребра (идем от вершин с меньшими номерами к большим по очереди), записываем результат в словарь, затем прописываем в матрицу смежные ребра. Код маркировки рёбер:

private static Pair<EdgeData[], HashMap<EdgeData, Integer>> markEdges(int m) {

EdgeData[] dict = new EdgeData[m];

HashMap<EdgeData, Integer> pairsToIndexes = new HashMap<>();

List<Node> sorted = getSorted();

int index = 0;

for (Node n : sorted) {

List<Node> neighbours = n.getNeighboursSorted();

for (Node neighbour : neighbours) {

EdgeData pair = new EdgeData(n.getNum(), neighbour.getNum(), -1);

if (n.getNum() < neighbour.getNum()) {

pairsToIndexes.put(pair, index);

dict[index] = pair;

pair.setIndex(index + 1);

index += 1;

} else {

Integer revertedIndex = pairsToIndexes.get(new EdgeData(neighbour.getNum(), n.getNum(), -1));

Integer realIndex = revertedIndex + m / 2;

pair.setIndex(realIndex + 1);

pairsToIndexes.put(pair, realIndex);

dict[realIndex] = pair;

}

}

}

return new Pair(dict, pairsToIndexes);

}

2) Записываем в json-файл матрицу и строку-словарь рёбер.

3) С помощью скрипта формируем LaTeX-результат.

4) Записываем результат в выбранный пользователем файл.

### **3.2.4 Описание реализации алгоритма вычисления функции Ихары на путях**

### Выберем какое-то остовное дерево T графа X. Обозначим ребра, которые в него не входят, как . Обозначим ребра из остовного дерева как .

### Зададим матрицу Z 2r x 2r где (i, j)-й элемент — комплексное число (произведение соответствующих w на пути от ребра к ребру через остовные ребра) если , и 0 иначе.

### Норма для примитивного пути P - .

### Формула функции Ихары на путях:

### Формула для вычисления функции в программе:

### Алгоритм:

### 1) Размечаем ребра как в пункте 3.2.4.

### 2) Находим любое остовное дерево, запоминаем ребра. Используется простой рекурсивный алгоритм DFS, специализированный под задачу. Если такого нет, возвращаем ошибку.

### 3) Составляем матрицу путей. Для каждой пары ребер, не входящей в остовное дерево, находим путь с помощью следующего рекурсивного алгоритма:

private static String recursiveFindPathBySpanningTree(

EdgeData curEdge,

EdgeData aimEdge,

HashMap<Integer, List<EdgeData>> spanningTree,

String res,

Set<Integer> visited

) {

if (curEdge.getTo() == aimEdge.getFrom()) {

return res + String.format("w\_{%d, %d}$", curEdge.getIndex(), aimEdge.getIndex());

}

if (curEdge.equals(aimEdge)) {

return "";

}

visited.add(curEdge.getTo());

for (EdgeData e : spanningTree.get(curEdge.getTo())) {

if (visited.contains(e.getTo()))

continue;

String curRes = recursiveFindPathBySpanningTree(

e,

aimEdge,

spanningTree,

res + String.format("w\_{%d, %d}", curEdge.getIndex(), e.getIndex()),

visited

);

if (curRes.length() > 1 && curRes.endsWith("$")) {

return curRes;

}

}

return "";

}

4) Записываются в json-файл матрица путей, маркировка ребер, ребра в остовном дереве и вне его.

5) Скриптом возвращается строка результата в формате LaTeX.

6) Выводится результат в выбранный пользователем файл.

### **3.2.5 Описание реализации алгоритма вычисления функции на путях**

Для произвольной вершины графа v зададим следующую функцию:

, где D(v) – множество длин всех путей из вершины v, которые не проходят через одно ребро дважды.

Алгоритм:

1) Пользователь выбирает вершину для вычисления с помощью контекстного меню.

2) С помощью специализированного обхода в глубину находятся все пути (символьные) из вершины:

private static void recursiveDFS(

Node curNode,

List<String> tokens,

List<String> paths,

Set<Edge> visited

) {

for (Edge e : curNode.getEdges()) {

Set<Edge> newVisited = new HashSet<>(visited);

if (e.getTextLength().contains("infty")) {

throw new ValidationException("There must be no infinities in distances");

}

if (visited.contains(e)) {

continue;

}

newVisited.add(e);

ArrayList<String> t = new ArrayList<>(tokens);

t.add(e.getTextLength());

paths.add(Parser.parseTexToSympy(t));

recursiveDFS(e.getNeighbour(curNode), t, paths, newVisited);

}

}

3) В json-файл записывается вершина и длины всех путей, формируется .tex файл.

### **3.2.6 Описание реализации алгоритма парсера**

Парсер LaTeX-длин ребер реализован с помощью алгоритма сортировочной станции, основанном на обратной польской нотации. Псевдокод алгоритма взят из источника [4] и переработан под специфику LaTeX выражений. Алгоритм переводит входную строку в обратную польскую нотацию, которую помещает в очередь, а затем вычисляет итоговое выражение. При этом парсер способен распознавать некорректные выражения. Команды, распознаваемые парсером: \infty, \sqrt[m]{x}, \frac{a}{b}, ^, \*, +, /. Помимо этого парсер умеет преобразовывать LaTeX-выражения в синтаксис SymPy. Это необходимо для вычисления функций с ребрами в виде корней из простых чисел. Полный код алгоритма доступен в документе, содержащем текст данной программы (класс Parser).

**3.3 Описание входных и выходных данных**

Входные данные:

- расположение узлов (с помощью mouse click)

- ребра и их форма (два mouse-click по узлам между которыми будет ребро

- форма изменяется с помощью anchor в центре ребра)

- длины ребер (mouse-click по метке на ребре и ввод с помощью клавиатуры)

- стартовая вершина для алгоритма вычисления функции на путях (right mouse-click и выбор во всплывшем контекстном меню)

- ранее сохраненный в программе граф в виде файла .graph

Выходные данные:

- файл в формате .tex или .txt с результатом вычисления функции

- файл в формате .graph с сохраненной моделью графа

## **3.4. Описание и обоснование выбора состава технических средств**

Для корректной работы программы необходим ПК с ОС Linux. Рекомендуются также следующие характеристики:

- минимум 500 MB оперативной памяти

- минимум 500 MB на жёстком диске

Для ввода данных необходимы клавиатура и мышь/тачпад.

## **3.5. Описание и обоснование выбора состава программных средств**

Графический интерфейс выполнен на языке Java с помощью платформы JavaFX, которая предоставляет широкий набор инструментов для его создания.

Вычисления функций выполнены на языке Python версии 3.6. Версия обусловлена использованием библиотеки для символьных вычислений SymPy, которая требует версию языка не ниже 3. Python и SymPy обеспечивают простую и быструю работу с символьными и матричными вычислениями, а также облегчают работу с LaTeX.

Так как Java-фреймворк Jython не поддерживает версию Python 3+, необходимую для работы с SymPy, не удалось достичь кросплатформенности приложения. Программа разработана под ОС Linux.

# **4.ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ**

[1] What are zeta functions of graphs and what are they good for? [Электронный ресурс] URL: https://math.ucsd.edu/~aterras/snowbird.pdf

[2] The magnitude of a graph [Электронный ресурс] URL: https://arxiv.org/pdf/1401.4623.pdf

[3] JavaFX animation [Электронный ресурс] URL: http:/​/​zetcode.com/​gui/​javafx/​animation/

[4] Алгоритм сортировочной станции [Электронный ресурс] // Wikipedia: [сайт]. URL: https:/​/​ru.wikipedia.org/​wiki/​%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BE%D1%87%D0%BD%D0%BE%D0%B9\_%D1%81%D1%82%D0%B0%D0%BD%D1%86%D0%B8%D0%

# **5. ПРИЛОЖЕНИЯ**

## **ПРИЛОЖЕНИЕ 1**

**ИСПОЛЬЗУЕМЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ**

Граф – упорядоченная пара (V,E), где V – непустое множество узлов(вершин), а E – множество рёбер.

Ребро – пара вершин (может быть упорядоченная или неупорядоченная)

Степень вершины – количество смежных с ней рёбер.

Связный граф – граф, для любых двух вершин a, b которого есть путь из a в b.

Неориентированный граф – граф, ребра которого представлены неупорядоченными парами.

Метрический граф – граф, где каждое ребро имеет длину.

Амплитуда – некоторая числовая величина точки, в квантовой механике характеризующая плотность вероятности найти точку на ребре.

Обратная польская нотация – форма записи математических выражений, в которых операнды стоят перед знаком операции.

## **ПРИЛОЖЕНИЕ 2**

**ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ**

|  |  |
| --- | --- |
| **Класс** | **Назначение** |
| Arrow | Визуализация вектора амплитуды точки |
| Distance | Длина ребра (отображение, ввод) |
| Edge | Ребро графа |
| Graph | Интерфейс, объединяющий бесконечные и конечные графы |
| InfiniteGraph | Абстрактный класс бесконечного графа |
| InfiniteManager | Класс, управляющий визуализацией на бесконечном графе и сменой типа графа |
| LatticeGraph | Граф-решетка |
| LineGraph | Граф, представленный бесконечной линией |
| Node | Вершина графа |
| Point | Точка, движущаяся по графу |
| Restorable | Интерфейс, отвечающий за умение объекта восстанавливаться |
| SimpleGraph | Граф, создаваемый пользователем |
| TexLabel | Непосредственное отображение длины ребра в виде LaTeX метки |
| Undoable | Интерфейс, помечающий, что объект реализует систему undo-redo |
| Visitable | Интерфейс, помечающий объект «посещаемым» для алгоритма DFS |
| Cache | Кэш, хранящий команды для undo-redo |
| Command | Интерфейс, обобщающий все команды, хранимые в undo-redo кэше |
| ChangeAllDistancesCommand | Команда сэтапа всех длин рёбер |
| ChangeDistCommand | Команда смены длины одного ребра |
| CreateCommand | Команда создания объекта |
| DeleteCommand | Команда удаления объекта |
| Controller | Отвечает за связь FXML интерфейса и кода |
| Drawer | Отвечает за отрисовку элементов на экране |
| EdgeContextMenu | Контекстное меню ребра |
| FileManager | Управление файловой системой |
| Filter | Отвечает за фильтрацию mouse-events |
| Formatter | Задает формат чисел с плавающей точкой |
| GIFMaker | Создание GIF-анимации визуализации |
| Invoker | Отвечает за «оборачивание» событий в команды и запись в кэш |
| Manager | Точка старта приложения и создания сцены |
| MenuManager | Управление контекстными меню |
| MyContextMenu | Кастомное контекстное меню |
| NodeContextMenu | Меню вершины |
| Parser | Парсер математических выражений |
| PopupMessage | Объект всплывающей подсказки |
| Visualizer | Отвечает за визуализацию распределения амплитуд |
| Token | Знак операции в парсере |
| Operation | Операция над двумя элементами |
| Function | Операция над одни элементом |

## **ПРИЛОЖЕНИЕ 3.**

**ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ МЕТОДОВ, ПОЛЕЙ И СВОЙСТВ**

**Класс Arrow**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| BASE\_LENGTH | private | double | Базовая длина стрелки | |
| curPos | private | Position | Текущая позиция стрелки на панели отрисовки | |
| triangle | private | Polygon | Треугольник стрелки (верхняя часть) | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| redrawArrow | Package private | void | double newVal | Перерисовывает стрелку там, чтобы ее длина соответствовала амплитуде |
| addArrow | Package private | void |  | Добавляет стрелку на экран |
| removeArrow | Package private | void |  | Убирает стрелку с экрана |
| setArrowTranslateX | Package private | void | double x | Задает сдвиг по оси x |
| setArrowTranslateY | Package private | void | double y | Задает сдвиг по оси y |
| drawTriangle | private | void | double newVal | Перерисовывает треугольник стрелки по амплитуде |
| getAbsMax | private | long |  | Возвращает максимальный модуль амплитуды, которая присутствует на экране |

**Класс Distance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| label | private | TexLabel | Объект, отображающий LaTeX | |
| input | private | TexField | Текстовое поле ввода длины | |
| value | private | double | Длина в виде вещественного числа | |
| curText | private | String | Длина в виде строки | |
| MAX\_LENGTH | public | int | Максимальная длина строки, которую может ввести пользователь | |
| isCalculated | private | boolean | Определяет, надо ли вычислять математические выражения в поле длины | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| setCalc | public | void |  | Меняет значение isCalculated |
| showInput | public | void |  | Прячет длину и выводит на экран поле ввода |
| showLabel | private | void |  | Прячет поле ввода и выводит на экран длину |
| show | Package private | void |  | Показывает длину полностью |
| hide | Package private | void |  | Прячет длину полностью |
| setDistance | public | void | String text, double val | Задает строковую и вещественную длины |
| getText | public | String |  | Геттер текстовой длины |
| calculate | public | void |  | Выводит вещественное значение длины |
| decalculate | public | void |  | Выводит введенное пользователем значение (если оно было выражением) |
| reset | Package private | void |  | Сбрасывает длину (в бесконечность) |
| isInfty | public | void |  | Является ли длина бесконечной |
| getValue | public | double |  | Возвращает вещественную длину |

**Класс Edge**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| n1, n2 | private | Node | Две вершины, смежные с ребром | |
| visited | private | boolean | Метка посещенности для DFS | |
| length | private | Distance | Объект длины ребра | |
| LABEL\_GAP | private | double | Отступ от ребра для лейбла длины | |
| color | private | Color | Базовый цвет ребра | |
| selectedColor | private | Color | Цвет ребра при выделении | |
| curColor | private | Color | Текущий цвет ребра | |
| pointsToProceed | private | ConcurrentHashMap<Integer, Point> | Точки, которые пришли в узлы на концах ребра | |
| canVisualize | private | ConcurrentHashMap<Integer, Boolean> | Можно ли выпустить новую точку по этому ребру из вершины с данным номером | |
| nearestCoords | private | HashMap<Node, double[]> | Координаты ближайшей к каждому ноду точки ребра | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| resetProceed | Package private | void |  | Сбрасывает нужные поля для новой визуализации |
| hide | Package private | void |  | Прячет ребро |
| show | Package private | void |  | Возвращает ребро на экран |
| select | public | void |  | Выбирает ребро (смена цвета) |
| deselect | public | void |  | Меняет цвет отмеченного ребра на обычный |
| getNodesNearest | public | double[] | Node n | Возвращает координаты ближайшего к узлу конца ребра |
| removePoint | private | void |  | Удаляет точку на ребре |
| handlePoint | Package private | PathTransition | Node n, int degree | Обрабатывает точку, которая пришла в узел, смежный с ребром |
| addToProceed | Package private | void | Node n, Point p | Добавляет пришедшую точку в hashmap обработки |
| setNodes | public | void | Node n1, Node n2 | Сэттер для смежных вершин |
| restore | public | void |  | Восстанавливает удаленное ребро |
| getNodes | public | Node[] |  | Геттер для смежных вершин |
| getNeighbour | public | Node | Node n | Возвращает соседа переданной вершины |
| connectNodes | public | void | Node n1, Node n2 | Соединяет два узла линией |
| getDistance | public | double | Double xPos, double yPos, double centerX, double centerY | Возвращает реальную длину (евклидово расстояние) |
| getStartCoordinates | public | Double[] | Double xPos, double yPos, double centerX, double century, double distance | Возвращает кординаты начала ребра на узле |
| create | public | boolean |  | Создает ребро |
| remove | public | void |  | Удаляет ребро |
| clone | public | Edge |  | Клонирует ребро |
| showLength | Package private | void |  | Показывает длину |
| hideLength | Package private | void |  | Прячет длину |
| resetLength | Package private | void |  | Сбрасывает значение длины |
| changeLength | public | void | String text, double val | Меняет значение длины |
| getLength | public | double |  | Геттер длины |
| getTextLength | Package private | String |  | Геттер текстовой длины |
| isVisited | public | boolean |  | Геттер посещенности |
| visit | public | void |  | Помечает ребро посещенным |
| unvisit | public | void |  | Снимает метку посещенности |
| relocateLabel | private | void |  | Перемещает лейбл длины |
| setHandlers | private | void |  | Устанавливает обработчики событий |

**Класс InfiniteGraph**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| nodes | private | | Collection<Node> | | | Список узлов | |
| EDGE\_LENGTH | public | | int | | | Длина ребра | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| getNodes | | protected | | Collection<Node> |  | | Геттер узлов |
| resetNodes | | public | | void |  | | Сбрасывает информацию о визуализации для всех вершин |
| visualize | | public | | void |  | | Запуск визуализации |
| stop | | public | | void |  | | Остановка визуализации |
| erase | | public | | void |  | | Удаление графа с экрана |

**Класс InfiniteManager**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| graph | private | InfiniteGraph | Текущий граф (из встроенных) | |
| currentType | private | Type (enum {Lattice, Line, Simple}) | Тип графа, который отображается в данный момент | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| init | public | void | Type type | Меняет тип графа на переданный |
| canEdit | public | boolean |  | Пользовательский ли граф сейчас отображается |
| visualize | public | void |  | Запуск визуализации |
| stop | public | void |  | Остановка визуализации |
| resetNodes | public | void |  | Сброс информации о визуализации |

**Класс LatticeGraph**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| ITERATIONS | private | | int | | | Число итераций построения | |
| NEIGHBOURS | private | | int | | | Число соседей каждого узла | |
| startRadius | private | | int | | | Радиус первой вершины | |
| counter | private | | int | | | Счетчик номеров вершин | |
| root | private | | Node | | | Корневая вершина | |
| startLength | private | | double | | | Стартовая длина ребра | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| drawLevel | | private | | void | Int curLevel, int curNodesSum, double curStart, double curLength | | Отрисовка очередного уровня решетки |
| redraw | | public | | Node |  | | Перерисовка |
| getAngle | | private | | Double[] | Double startX, double startY, double angle, double length | | Возвращает координаты центра узла, который отклонен от переданного на угол |
| addNode | | private | | Node | Double x, double y, double radius | | Добавление нового узла |
| addEdge | | private | | void | Node n1, Node n2 | | Соединяет узлы ребром |
| erase | | public | | void |  | | Стирает граф |
| stop | | public | | void |  | | Остановка визуализации |
| visualize | | Public | | void |  | | Старт визуализации |

**Класс LineGraph**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| NUM\_OF\_NODES | private | | int | | | Число вершин в линии | |
| START\_X | private | | double | | | Координата x первого узла | |
| START\_Y | private | | double | | | Координата y первого узла | |
| graph | private | | Group | | | Группа всех ребер и узлов | |
| transition | private | | PathTransition | | | Движение графа | |
| ft | private | | FadeTransition | | | Opacity первого нода | |
| edgeFt | private | | FadeTransition | | | Opacity первого ребра | |
| line | private | | LineTo | | | Координаты конца анимации движения | |
| move | private | | MoveTo | | | Координаты начала анимации движения | |
| path | private | | Path | | | Линия движения | |
| hero | private | | Point | | | Точка для визуализации | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| stop | | public | | void |  | | Остановка визуализации |
| addNode | | private | | Node | Node prevNode | | Добавление нового узла в линию |
| visualize | | Public | | void |  | | Старт визуализации |
| erase | | public | | void |  | | Удаление графа с экрана |
| animate | | private | | void |  | | Запуск анимации |

**Класс Node**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| RADIUS | public | double | Стандартный радиус вершины | |
| num | private | int | Номер вершины | |
| visited | private | boolean | Метка посещенности | |
| edges | private | ArrayList<Edge> | Смежные ребра | |
| color | private | Color | Базовый цвет | |
| selectedColor | private | Color | Цвет при выделении | |
| curColor | private | Color | Текущий цвет | |
| processed | Package private | BooleanProperty | Ждет ли вершина точки в данный момент | |
| amplitudesSum | private | double | Сумма амплитуд которые пришли в вершину в данный момент | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| getCircle | public | Circle |  | Возвращает круг, которым отрисована вершина |
| getAmplitudesSum | Package private | double |  | Возвращает текущую сумму амплитуд в вершине |
| increaseAmplitudesSum | Package private | void | Double amplitude | Добавляет амплитуду к текущей сумме |
| resetNode | Package private | void |  | Сброс информации для визуализации |
| checkMinEdge | Package private | void |  | Проверяет длины смежных рёбер на минимальность |
| getNum | public | int |  | Возвращает номер вершины |
| select | public | void |  | Выбирает вершину как стартовую |
| deselect | public | void |  | Снимает отметку выбора |
| getNeighbours | Package private | Set<Node> |  | Возвращает коллекцию соседей |
| getEdges | public | ArrayList<Edge> |  | Возвращает смежные ребра |
| renewNum | Package private | void | int num | Обновляет номер вершины |
| rescaleX | Package private | void | Double scale | Перемещает узел по x в соответствии с переданным параметром |
| rescaleY | Package private | void | Double scale | Перемещает узел по y |
| getText | public | Text |  | Геттер текстового объекта |
| addEdge | Package private | Boolean | Node neighbour, Edge edge | Добавляет смежное ребро |
| removeNeighbour | Package private | void | Node n | Удаляет соседа |
| toString | public | String |  | Перегрузка метода |
| relocateCircleCenter | private | void | Double x, double y | Перемещает вершину в переданную точку |
| fixPosition | public | void | Double xPos, double yPos | Фиксирует позицию нода после перемещения |
| remove | public | void |  | Удаляет узел со сцены |
| create | public | boolean |  | Создает узел |
| isVisited | public | boolean |  | Посещен ли узел |
| visit | public | void |  | Помечает посещенным |
| unvisit | public | void |  | Снимает отметку посещенности |
| showLengths | Package private | void |  | Показывает длины смежных ребер |
| hideLengths | Package private | void |  | Прячет длины смежных ребер |
| restore | public | void |  | Восстанавливает узел |
| handleEdges | private | void | Consumer<Edge> handler | Обработчик рёбер в dfs |
| recalculateEdges | private | void |  | Заново соединяет вершину с ребрами |
| checkBoundsCrossed | private | Boolean[] | MouseEvent event | Проверка на то, пересекает ли вершина границы отрисовки |
| setText | private | void |  | Сеттер текста |
| setHandlers | private | void |  | Устанавливает нужные обработчики событий |
| handlePoints | private | void |  | Обрабатывает все ребра и запускает новые анимации точек |

**Класс Point**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| RADIUS | Package private | int | Базовый радиус точки | |
| SHIFT | private | int | Сдвиг числового значения амплитуды от точки | |
| BASE\_COLOR | private | Color | Базовый цвет | |
| amplitude | private | Double | Числовая амплитуда | |
| destination | private | Node | Вершина, к которой движется точка | |
| pathTransition | private | PathTransition | Анимация | |
| edge | private | Edge | Ребро, на которой точка | |
| line | private | LineTo | Конечные координаты анимации | |
| path | private | Path | Путь анимации | |
| move | private | MoveTo | Начальные координаты анимации | |
| numAplitude | private | Text | Объект числового значения амплитуды | |
| arrow | private | Arrow | Объект вектора амплитуды | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| getAmplitude | public | String |  | Геттер амплитуды |
| showAttributes | public | void |  | Показывает стрелку и число, если нужно |
| showNumbers/showArrows/showColour | public | void |  | Показывает число/стрелку/цвет |
| hideNumbers/hideArrows/hideColour | public | void |  | Скрывает число/стрелку/цвет |
| hideEnabled | public | void |  | Скрывает те атрибуты, которые показаны |
| setDestination | Package private | void | Node n | Задает новый пункт назначения |
| changeAmplitude | Package private | void | Int degree | Меняет амплитуду |
| setAmplitude | Package private | void | Int degree | Задает амплитуду |
| removePath | public | void |  | Удаляет анимацию из визуализатора |
| startPath | public | PathTransition | Double[] start, double[] end, double startEdge | Создает объект анимации |
| setPointToEdge | private | void |  | Присваивает точку ребру |
| calculateInterpolation | private | Color |  | Вычисляет цвет точки |
| setBindings | private | void |  | Задает обработчики событий |
| updateInfo | private | void | Node toCheck | Апдейт общей информации после задания амплитуды |

**Класс SimpleGraph**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| MAX\_SIZE | public | | int | | | Максимальное количество узлов, которое можно создать | |
| nodes | private | | ArrayList<Node> | | | Список всех вершин графа | |
| instance | private | | SimpleGraph | | | Синглтон | |
| dfsStack | private | | Stack<Node> | | | Стэк для DFS | |
| showDistances | private | | boolean | | | Показываются ли длины ребер | |
| startEdge | private | | Edge | | | Ребро для старта визуализации | |
| startNode | private | | Node | | | Вершина для старта визуализации | |
| curMinEdge | private | | double | | | Минимальная длина ребра в текущем графе | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| getStartEdge | | public | | Edge |  | | Геттер стартового ребра |
| getStartNode | | public | | Node |  | | Геттер стартового узла |
| setStartEdge | | public | | void | Edge start | | Сеттер стартового ребра |
| setStartNode | | public | | void | Node start | | Сеттер стартового узла |
| setMin | | public | | void | Double pretender | | Проверяет и обновляет минимальную длину ребра |
| getCurMinEdge | | public | | double |  | | Геттер минимальной длины ребра |
| getInstance | | public | | SimpleGraph |  | | Геттер инстанса |
| getSize | | public | | int |  | | Геттер числа узлов |
| areDistancesShown | | public | | boolean |  | | Показываются ли длины ребер |
| hideGraph | | public | | void |  | | Прячет граф |
| showGraph | | public | | void |  | | Показывает граф |
| addNode | | Package private | | void | Node node | | Добавляет новый узел |
| removeNode | | Package private | | void | Node circle | | Удаляет узел |
| refreshLabels | | Package private | | void | Node circle | | Обновляет значения лейблов после удаления переданного узла |
| getEdgesAndDistances | | public | | HashMap<Edge, Pair<String, Double>> |  | | Метод для операции setAll у инвокера |
| clearGraph | | public | | void |  | | Очищает список узлов |
| rescale | | Public | | void | Char axis, double oldVal, double newVal | | Перемещает граф после ресайза сцены |
| resetNodes | | public | | void |  | | Сбрасывает информацию о визуализации для всех узлов |
| setLengths | | public | | void |  | | Показывает длины всех ребер |
| hideLengths | | Public | | void |  | | Прячет все длины |
| resetDistances | | public | | void |  | | Сбрасывает длины в бесконечность |
| changeDistances | | public | | void | String input | | Меняет длины ребер на переданное |
| visualizeAmplitudes | | public | | void |  | | Делает проверку графа на корректность и запускает визуализацию |
| DFS | | private | | void | Consumer<Node> handler | | Поиск в глубину из одного узла |
| runDFS | | private | | int | Consumer<Node> handler | | Поиск в глубину (применяет хендлер ко всем узлам) |
| resetDFS | | private | | void |  | | Сбрасывает метки посещенности всех узлов |

**Класс TexLabel**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| DEFAULT | public | | String | | | Дефолтное значение лейбла длины | |
| gc | private | | FXGraphics2D | | | Объекты для отображения LaTeX | |
| icon | private | | TeXIcon | | |
| input | private | | TextField | | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| setText | | Package private | | String | String text | | Задает текст для LaTeX объекта |

**Класс Cache**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| CAPACITY | private | int | Вместимость кэша | |
| stack | private | Command[] | Список команд в стеке | |
| curCapacity | private | Int | Текущая заполненность стека | |
| pointer | private | Int | Указатель для undo комманд | |
| redoPointer | private | int | Указатель для redo комманд | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| push | public | Command | Command elem | Добавляет команду в стек |
| pop | public | Command |  | Удаляет последнюю команду (сдвигом указателя) |
| getNext | public | Command |  | Возвращает следующую за последней команду |
| getCurrent | public | Command |  | Возвращает текущую команду |

**Интерфейс Command (для всех классов Command методы повторяются)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| undo | public | void |  | Отменяет последнюю команду |
| execute | boolean | void |  | Воспроизводит следующую команду в стеке |

**Класс ChangeDistCommand**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| dist | private | Distance | Объект длины, над которым совершалась операция |
| oldDist | private | String | Старое значение длины (текст) |
| oldVal | private | Double | Старое значение длины (число) |
| newDist | private | String | Новое значение длины (текст) |
| newVal | private | Double | Новое значение длины (число) |

**Класс CreateCommand**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| elem | private | Undoable | Элемент, который создавался |

**Класс DeleteCommand**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| elem | private | Undoable | Элемент, который удалялся |
| cachedEdges | private | ArrayList<Edge> | Сохраненные ребра, если удалялся узел |

**Класс ChangeAllDistancesCommand**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| commonValue | private | Double | Введенная длина (число) |
| initialized | private | boolean | Были ли инициализированы длины |
| input | private | String | Введенная длина |
| oldVals | private | HashMap<Edge, Pair<String, Double>> | Старые длины ребер |

**Класс Controller**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| Все поля соответствуют элементам интерфейса. | | | | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| setIcons | public | void |  | Подгрузка иконок кнопок |
| setButtons | private | void |  | Привязывает анимацию к кнопкам |
| openFile | private | void |  | Диалог открытия файла |
| saveUnchanged | Package private | void |  | Сохранение графа |
| addListeners | private | void |  | Привязывает к сцене нужные обработчики событий |
| discardAndOpen | Package private | void |  | Открывает файл и сбрасывает несохраненные данные |
| hideDialog | Package private | void |  | Прячет файловый диалог |
| initialize | Package private | void |  | Инициализация сцены |
| createNode | Package private | void | MouseEvent event | Создание узла по клику |
| clearWorkingArea | Package private | void |  | Очищает область отрисовки |
| undoAction | Package private | void |  | Отменяет действие |
| redoAction | Package private | void |  | Повторяет отмененное действие |
| showDist | Package private | void |  | Показывает длины ребер |
| hideDist | Package private | void |  | Скрывает длины ребер |
| resetDist | Package private | Void |  | Сбрасывает длины ребер в бесконечность |
| changeDist | Package private | void |  | Меняет длины ребер |
| visualizeAmplitudes | Package private | void |  | Запуск анимации |
| createGIF | Package private | void |  | Создание .gif файла |
| stopVisualizing | Package private | void |  | Остановка анимации |
| setTime | Package private | void |  | Задает время для .gif |
| drawSimple | Package private | void |  | Выбор пользовательского графа |
| drawLine | Package private | void |  | Выбор графа-линии |
| drawLattice | Package private | void |  | Выбор графа-решетки |

**Класс Drawer**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| BOUNDS\_GAP | public | int | Минимальное расстояние от края области отрисовки до узла | |
| NODE\_TEXT | public | String | Стиль для текста узла | |
| instance | private | Drawer | синглтон | |
| pane | private | AnchorPane | Панель отрисовки | |
| dialog | private | StackPane | Файловый диалог | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| getInstance | public | Drawer |  | Геттер синглтона |
| clear | public | void |  | Очищает область отрисовки |
| takeSnap | public | WritableImage |  | Делает снимок экрана |
| removeElement | public | void |  | Удаляет элемент из области отрисовки |
| setFocus | Package private | void |  | Фокус на область отрисовки |
| setPane | Package private | void |  | Сеттер панели отрисовки |
| enableDialog | public | void |  | Выводит на экран файловый диалог |
| setMoveHandler | Package private | void | EventHandler h | Добавляет хендлер на панель |
| removeMoveHandler | Package private | void |  | Удаляет хендлер |
| addElem | public | void |  | Добавляет элемент на панель отрисовки |
| removePoints | Package private | void |  | Удаляет с экрана все точки |
| getBounds | public | Bounds |  | Возвращает границы отрисовки |
| drawNode | Package private | Node | MouseEvent ev | Добавление узла в область клика |
| drawInfiniteNode | public | Node | Double xPos, double yPos, int num, double radius, boolean needText | Добавление узла для бесконечного графа |
| createLayout | private | Node | Double xPos, double yPos | Отрисовка модели узла |
| checkBounds | private | Double[] | Double xPos, double yPos | Проверка на выход за границы отрисовки |

**Класс EdgeContextMenu, NodeContextMenu**

Добавляют хендлеры ребрам и нодам соответственно в конструкторе.

**Класс FileManager**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| curFile | private | File | Текущий файл | |
| fileChooser | private | FileChooser | Диалог для выбора файла .graph | |
| gifChooser | private | FileChooser | Диалог для выбора файла .gif | |
| mainStage | private | Stage | Инстанс главной сцены | |
| noSave | private | BooleanProperty | Требуется ли сохранение | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| getGifFile | public | File |  | Открывает диалог сохранения .gif файла |
| setStage | public | void | Stage stage | Сеттер сцены |
| getDisable | Package private | BooleanProperty |  | Геттер необходимости сохранения |
| setNoSave | Package private | Boolean val | Boolean val | Сеттер необходимости сохранение |
| isSaveNeeded | Package private | boolean |  | Нужно ли сохранение |
| save | Package private | void |  | Сохраняет текущий файл |
| saveAs | Package private | void |  | Сохранение файла в конкретный файл выбранный пользователем |
| open | Package private | void |  | Открывает текущий файл |
| convertGraph | private | void | File file | Сериализует текущий граф для сохранения |

**Класс Filter**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| SELECTED\_BUTTON | private | String | Стиль для выбранной кнопки | |
| UNSELECTED\_BUTTON | private | String | Стиль кнопки, которая не выделена | |
| dragging | private | boolean | Осуществляется ли dragging в текущий момент | |
| edgeStarted | private | boolean | Создается ли новое ребро | |
| editing | private | boolean | Происходит ли редактирование графа | |
| pretender | private | Node | Узел, из которого начато ребро | |
| edgePretender | private | Edge | Ребро, которое еще не соединено со вторым узлом | |
| CURSOR\_GAP | private | int | Промежуток между концом начатого ребра и курсором | |
| dragFilter | public | EventFilter<MouseEvent> | Обработчик всех событий dragging в области отрисовки | |
| clickFilter | public | EventHandler<MouseEvent> | Обработка всех событий кликов в области отрисовки | |
| edgeMoveHandler | private | EventHandler<MouseEvent> | Обработчик движений мыши при создании нового ребра | |
| buttonEnterHandler | Package private | EventHandler<MouseEvent> | Обработчик попадания курсора на кнопку | |
| buttonExitHandler | Package private | EventHandler<MouseEvent> | Обработчик выхода курсора за область кнопки | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| isEdgeStarted | public | boolean |  | Редактируется ли новое ребро |
| isEditing | public | boolean |  | Происходит ли редактирование области отрисовки |
| endEdit | public | void |  | Закончить редактирование |
| removeStartedEdge | private | void |  | Удаляет начатое ребро, если его соединили с пустотой |

**Класс Formatter**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| formatter | private | DecimalFormat | Определяет число знаков после запятой | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| format | Public | String | Double val | Форматирует переданное число |

**Класс GIFMaker**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| images | private | List<WritableImage> | Список скринов для записи в .gif | |
| DEFAULT\_TIME | public | int | Дефолтная длительность .gif | |
| timer | private | Timer | Таймер анимации | |
| DELAY\_TIME | prrivate | int | Задержка между скринами графа | |
| out | private | ImageOutputStream | Поток записи файла | |
| time | private | int | Время записи .gif | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| setTime | public | String | String input | Сеттер времени .gif |
| createGif | public | void |  | Запуск записи |
| takeSnapshots | public | void |  | Запускает тред, который делает снимки экрана |
| stopTimer | public | void |  | Остановка таймера |
| isTimeDefault | public | boolean |  | Дефолтное ли время установлено |
| specifyMetaData | private | void | IIOMetadata metadata | Задает параметры класса, создающего .gif |
| getNode | private | IIOMetadata | IIOMetadataNode rootNode, String nodeName |  |
| specifyOutput | private | void |  | Задает файл для записи |

**Класс Invoker**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| commands | private | Cache | Кэш комманд | |
| instance | private | Imvoker | Синглтон | |
| toUndo | private | Command | Команда, которую надо отменить | |
| toRedo | private | Command | Команда, которую надо сохранить | |
| lastSaveCommand | private | Command | Последняя команда, на которой граф сохранялся | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| getInstance | public | Invoker |  | Геттер синглтона |
| reset | Package private | void |  | Сброс кеша |
| renewLastCommand | Package private | void |  | Обновляет последнюю сохраненную команду |
| checkLastCommand | Package private | void |  | Проверяет, нужно ли сохранение текущего графа |
| createElement | public | void | Undoable el | Запись команды создания элемента |
| changeAllDistances | public | void | String input | Запись команды изменения всех длин |
| deleteElement | public | void | Distance d, String text, double val | Запись команды удаления элемента |
| undoLast | Package private | void |  | Отменяет последнюю команду |
| redoLast | Package private | void |  | Повторяет отмененную ранее команду |

**Класс Manager**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| main | public | void | String args[] | Входная точка приложения |
| start | public | void | Stage stage | Инициализация главной сцены |

**Класс MenuManager**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| edgeMenu | private | EdgeContextMenu | Контекстное меню ребра | |
| nodeMenu | private | NodeContextMenu | Контекстное меню вершины | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| getEdgeMenu | public | EdgeContextMenu |  | Геттер контекстного меню ребра |
| getNodeMenu | public | NodeContextMenu |  | Геттер контекстного меню вершины |

**Класс MyContextMenu**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Методы | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Аргументы** | **Назначение** |
| show | public | | void | | | javafx.scene.Node node, double x, double y | Показывает контекстное меню |
| bindElem | public | | void | | | Javafx.scene.Node el | Привязывает элемент к меню |
| Поля | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Назначение** | | |
| deletion | | private | | MenuItem | Пункт удаления в меню | | |
| elem | | protected | | Undoable | Элемент, к которому привязано меню | | |

**Класс Token**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| val | private | String | Знак операции |

**Класс Operation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| priority | private | int | Приоритет операции | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| execute | Package private | Double | Double x, double y | Выполняет операцию над переданными значениями |

**Класс Function**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| data | Package private | | double | | | Степень корня | |
| fractionCounter | Package private | | int | | |  | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| execute | | Package private | | double | Double x | | Применяет функцию к переданному значению |

**Класс Parser**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| operations | private | | HashMap<Character, Operation> | | | Список доступных парсеру операций | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| parseDistance | | public | | double | String input | | Парсит переданную строку в число |
| createPolandNotation | | private | | void | ArrayDeque<Token> queue, ArrayDeque<Token> stack, String input | | Создает польскую нотацию из строки |

**Класс PopupMessage**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| popup | private | | Label | | | Текст всплывающей подсказки | |
| ft | private | | FadeTransition | | | Анимация, растворяющая подсказку | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| setPopup | | public | | void | Label label | | Инициализация анимации и лейбла |
| showMessage | | public | | void | String mes | | Вызывает всплывающую подсказку |
| fixMessage | | public | | void | String mes | | Фиксирует сообщение на экране (без анимации) |
| unfixMessage | | Public | | void |  | | Открепляет зафиксированное сообщение |

**Класс Visualizer**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| MAX\_POINTS | private | | int | | | Максимальное число точек, которое может быть отрисовано | |
| GAP | private | | int | | | Время ожидания точек в узле | |
| threadPool | private | | ExecutorService | | | Треды, которые запускают ожидание точек в ноде | |
| animations | private | | HashSet<PathTransition> | | | Список всех активных анимаций | |
| isRunning | private | | boolean | | | Идет ли визуализация | |
| enabledGIF | private | | boolean | | | Записывается ли .gif | |
| numeric | private | | boolean | | | Отображаются ли числа амплитуд | |
| colour | private | | boolean | | | Идет ли отрисовка амплитуды цветом | |
| arrows | private | | boolean | | | Отображаются ли стрелки амплитуд | |
| curNumOfPoints | private | | IntegerProperty | | | Текущее количество точек на сцене | |
| needStartPeriod | private | | boolean | | | Нужно ли начинать новый период | |
| startTime | private | | long | | | Время старта предыдущего периода | |
| periodTime | private | | long | | | Время периода | |
| timeout | private | | int | | |  | |
| globalStart | private | | Node | | | Вершина, в которой начался период | |
| observer | private | | ChangeListener<Number> | | | Отслеживает начало периода | |
| lowerBound | private | | LongProperty | | | Текущее минимальное значение амплитуды | |
| upperBound | private | | LongProperty | | | Текущее максимальное значение амплитуды | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| enableGif | | public | | void | Boolean enable | | Сеттер записи .gif |
| runTask | | public | | void | Task t | | Запускает задачу в локальном тред пуле |
| setNumeric | | public | | void | Boolean val | | Управление отображением числовой амплитуды |
| isNumeric | | public | | boolean |  | | Геттер числовой амплитуды |
| setColour | | Package private | | void | Boolean val | | Управление отображением амплитуды цветом |
| isColoured | | public | | boolean |  | | Геттер цветовой амплитуды |
| setArrows | | Package private | | void | Boolean val | | Управление отображением стрелок |
| isArrows | | public | | boolean |  | | Геттер стрелок векторов |
| increasePoints | | public | | void |  | | Увеличение счетчика точек в области отрисовки |
| decreasePoints | | public | | void |  | | Уменьшение счетчика точек в области отрисовки |
| checkOOM | | public | | boolean |  | | Проверка на превышение числа точек |
| getLowerBound | | public | | LongProperty |  | | Минимальная амплитуда на сцене |
| getUpperBound | | public | | LongProperty |  | | Максимальная амплитуда на сцене |
| checkMinMaxAmplitudes | | public | | void | Double val, boolean pretender, Node destination | | Проверка на то, нужно ли менять цветовой отрезок после появления новой амплитуды |
| bindBounds | | Package private | | void | Label lower, Label upper | | Привязывает значения лейблов к соответствующим свойствам |
| startVisualization | | public | | void | Edge startEdge, Node startNode | | Запуск визуализации |
| runLineVisualization | | public | | void | Point p, Runnable supplier | | Запуск визуализации на бесконечной линии |
| stopLineVisualization | | public | | void |  | | Остановка визуализации на линии |
| addPath | | public | | void | PathTransition p | | Добавляет анимацию в список активных |
| removePath | | Public | | void | PathTransition p | | Удаляет анимацию из списка |
| stopVisualization | | public | | void |  | | Останавливает визуализацию |
| isRunning | | public | | boolean |  | | Запущена ли визуализация |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | | | | | | |
| Номера листов (страниц) | | | | | | Всего листов (страниц в докум.) | | № документа | | Входящий № сопроводительного докум. и дата | | Подп. | | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированых |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |
|  |  |  |  |  |  | |  | |  | |  | |  | |

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**