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 41**

**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 реализации алгоритма вычисления функции магнитуд** 5](#_Toc39538997)

**3.2.2 Описание реализации алгоритма вычисления функции Ихары на вершинах**7

**3.2.3 Описание реализации алгоритма вычисления функции Ихары на ребрах**8

**3.2.4 Описание реализации алгоритма вычисления функции Ихары на путях**10

**3.2.5 Описание реализации алгоритма вычисления функции на путях**11

**3.2.6 Описание реализации алгоритма парсера**12

**3.3 Описание входных и выходных данных** [1](#_Toc39539000)4

[**3.4 Описание и обоснование выбора состава технических средств** 1](#_Toc39539000)4

[**3.5 Описание и обоснование выбора состава программных средств** 1](#_Toc39539001)4

[**4. ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ** 1](#_Toc39539002)5

[**5. ПРИЛОЖЕНИЯ** 16](#_Toc39539003)

[**ПРИЛОЖЕНИЕ 1** 1](#_Toc39539004)6

[**ПРИЛОЖЕНИЕ 2** 1](#_Toc39539005)7

[**ПРИЛОЖЕНИЕ 3.**](#_Toc39539006) 20

# **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**

Для того чтобы было возможно записывать длины в виде математических выражений был реализован алгоритм сортировочной станции [4] и переработан под LaTeX. Алгоритм основан на преобразовании выражения в обратную польскую нотацию. Парсер умеет распознавать следующие действия: ^, \*, +, /, \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. Необходимы также мышь или тачпад для ввода данных для построения графа. Рекомендуется также иметь не менее 500МБ памяти на жестком диске и не менее 500 МБ оперативной памяти.

## **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 – множество вершин V и попарно соединяющих их линий (рёбер) E.

Степень вершины – число выходящих из нее рёбер.

Неориентированное ребро — ребро, представленное неупорядоченной парой вершин.

Ориентированное ребро — ребро, представленное упорядоченной парой вершин.

Связный граф – граф, где между любой парой вершин есть хотя бы один путь.

Неориентированный граф — граф, все ребра которого неориентированные.

Метрический граф – граф, где каждое ребро имеет длину.

Матрица — таблица элементов из строк и столбцов.

Остовное дерево графа — подграф, имеющий |V| вершин и |V|-1 рёбер.

DFS, поиск в глубину, обход в глубину — метод обхода графа, идущий вглубь пока это возможно.

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

## **ПРИЛОЖЕНИЕ 2**

**ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ**

|  |  |
| --- | --- |
| **Класс** | **Назначение** |
| Anchor | Якорь, с помощью которого изменяется дуга ребра |
| AlgorithmService | Сервис, отвечающий за все вычисления связанные с графами |
| CustomFunctionDto | Задает json-формат данных для функции на путях |
| CustomFunctionService | Сервис, отвечающий за вычисление функции на путях |
| EdgeDistance | Объект, отвечающий за ввод и отображение длины ребра. Связан с ребром графа |
| Edge | Ребро графа |
| EdgeData | Data-класс, содержащий информацию о ребре |
| Graph | Модель графа, создаваемого пользователем |
| IharaDto | Задает json-формат данных для функции Ихаря на вершинах |
| IharaEdgeDto | Задает json-формат данных для функции Ихары на ребрах |
| IharaPathDto | Задает json-формат данных для функции на путях |
| IharaZetaFunctionService | Сервис, отвечающий за вычисление функций Ихары |
| MagnitudeService | Сервис, отвечающий за вычисление функции магнитуд |
| Node | Вершина графа |
| PythonService | Сервис, отвечающий за взаимодействие Java-классов и Python-скриптов |
| Restorable | Интерфейс, свойство восстанавливать объект (отрисовка и свойства) |
| TexLabel | LaTeX-метка длины ребра |
| Undoable | Интерфейс, свойство системы undo-redo |
| Visitable | Интерфейс, необходим для пометок в обходе в глубину |
| Cache | Кэш, хранящий команды для undo-redo |
| Command | Интерфейс команды в кэше |
| SetAllLengthsCommand | Команда сэтапа всех длин рёбер |
| SetSingleLengthCommand | Команда смены длины одного ребра |
| CreateCommand | Команда создания объекта |
| DeleteCommand | Команда удаления объекта |
| InterfaceController | Связывает объекты UI с функциями в коде |
| DrawingAreaController | Отрисовывает вершины и добавляет/удаляет объекты в область отрисовки |
| EdgeContextMenu | Контекстное меню ребра |
| FileManager | Работа с файлами, загрузка/выгрузка |
| EventFilter | Фильтрует все события (клики, перетаскивания и тд) |
| GUIStarter | Обертка для запуска приложения (нужна Мавену) |
| Invoker | Оборачивает кэшируемые события в объекты команд и добавляет в кэш |
| IsAlreadyVisitedException | Исключение при нахождении цикла в алгоритме проверки графа на цикличность |
| Manager | Создает и запускает графический интерфейс |
| MenuManager | Управление контекстными меню |
| CommonContextMenu | Контекстное меню, объединяющее в себе общие верты меню всех элементов |
| NodeContextMenu | Контекстное меню вершины |
| Parser | Парсер LaTeX-выражений (вычисление, перевод в формат SymPy) |
| PopupMessage | Окно с уведомлением пользователю (всплывающая подсказка) |
| Token | Знак операции в парсере |
| ValidationException | Исключение при валидации графа перед вычислением функции |
| Operation | Операция над двумя числами в парсере |
| Function | Функция, наследуется от операции и принимает одно число (реализована для корня) |

**Файлы ihara.py, custom.py, ihara\_edge.py, ihara\_path.py, magnitude.py, utils.py реализуют вычисления соответствующих функций с помощью SymPy и вывод результата в формате .tex строки.**

## **ПРИЛОЖЕНИЕ 3.**

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

**Класс AlgorithmService**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| dfsStack | private | Stack<Node> | Стэк вершин для поиска в глубину | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| findAllPaths | public | List<String> | Node node | Находит все пути из заданной вершины для функции на путях |
| findAllMinDistances | public | String[][] |  | Находит матрицу минимальных расстояний для магнитуды графа |
| findPathMatrix | public | IharaPathDto |  | Находит матрицу путей для Изары на путях |
| findEdgeMatrix | public | IharaEdgeDto |  | Находит матрицу ребер для Ихары на ребрах |
| findAdjacencyMatrix | public | int[][] |  | Поиск матрицы смежности для невзвешенной функции Ихары на вершинах |
| findWeightedMatrix | public | String[][] |  | Поиск матрицы весов для взвешенной функции Ихары на вершинах |
| findDiagonalMatrix | public | int[][] |  | Поиск диагональной матрицы графа для Ихары на вершинах |
| hasCycles | public | boolean |  | Проверка на циклы |
| runDFS | public | int | Consumer<Node> handler | Запускает обход в глубину из вершины |
| recursiveFindPathBySpanningTree | public | String | EdgeData curEdge, EdgeData aimEdge, HashMap<Integer, List<EdgeData> spanningTree,  String res,  Set<Integer> visited | Поиск пути из одного ребра в другое по остовному дереву для Ихары на путях |
| getEdgesDictString | private | String | EdgeData[] dict | Строка со словарем ориентированных ребер |
| getSorted | private | List<Node> |  | Отсортированные вершины графа (по номеру) |
| markEdges | private | Pair<EdgeData[], HashMap<EdgeData, Integer>> | Int m | Ориентирует ребра для Ихары на ребрах и путях |
| findSpanningTree | private | Set<Integer> | List<Node> nodes, HashMap<EdgeData, Integer> dict | Поиск остовного дерева |
| findMinDistance | private | void | Node n, String[] distances | Поиск минимальных расстояний для магнитуды графа, алгоритм Дейкстры |
| hasCycle | private | void | Node n, Node parent | Проверка вершины на наличие в ней цикла |
| recursiveDFS | private | void | Node curNode, List<String> tokens, List<String> paths, Set<Edge> visited | Рекурсивный обход в глубину для поиска всех путей из вершины |
| DFS | private | void | Consumer<Node> handler | Обход в глубину из текущей вершины |

**Класс Anchor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| RADIUS | private | double | Радиус круга | |
| BASE\_COLOR | private | Color | Цвет границы | |
| LIGHT\_COLOR | private | Color | Цвет заполнения | |
| TRANSPARENT | private | Color | Прозрачный цвет | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| enableDraggingEdgeOnDraggingAnchor | private | void | BiConsumer<Double, Double> anchorManager | Задает коллбэки: при перетаскивании якоря меняется форма ребра |
| setNewCoordinatesSafe | public | void | double newX, double newY | Контролирует чтобы якорь и ребро не пересекли границы отрисовки |
| show | public | void |  | Показывает якорь |
| hide | public | void |  | Прячет якорь |

**Класс EdgeDistance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| texLabel | private | TexLabel | Библиотечный лейбл-картинка для LaTeX-выражения | |
| inputField | private | TexField | Окно ввода | |
| value | private | double | Значение (вещественное число) | |
| textValue | private | String | Значение (невычисленная строка-выражение) | |
| MUST\_CALCULATE | private | boolean | Текстовое или вещественное значение выводить | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| showInput | public | void |  | На экране окно ввода |
| showLabel | public | void |  | На экране введенное выражение в виде метки на ребре |
| toScreen | public | void |  | Выводит длину на экран |
| fromScreen | public | void |  | Убирает длину с экрана |
| setDistance | public | void | String text, double val | Задает значения (вычисленное и символьное) |
| showText | public | void |  | Показывается числовое значение |
| showNumeric | public | void |  | Показывается исходное значение (символьное) |
| toInfty | public | void |  | Сетит длину в бесконечность |

**Класс Edge**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| startNode | private | Node | Вершина откуда выходит ребро | |
| endNode | private | Node | Вершина куда входит ребро | |
| length | private | EdgeDistance | Объект длины ребра | |
| anchor | private | Anchor | Якорь ребра, контролирующий отрисовку | |
| DISTANCE\_LABEL\_GAP | private | double | Отступ от ребра для лейбла длины | |
| color | private | Color | Базовый цвет ребра | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| getNodesNearest | public | double[] | Node n | Возвращает координаты ближайшего к узлу конца ребра |
| restore | public | void |  | Восстанавливает удаленное ребро |
| getNeighbour | public | Node | Node n | Возвращает соседа переданной вершины |
| connectNodes | public | void | Node startNode, Node endNode | Соединяет два узла дугой |
| getDistance | public | double | Double xPos, double yPos, double centerX, double centerY | Расстояние Евклида по заданным координатам |
| 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 |
| changeLengthValue | public | void | String text, double val | Задает новую длину |
| relocateAnchor | private | void | Double centerX, double centerY, double xCoef, double yCoef | Перемещает якорь при изменении длины и положения ребра |
| createAnchor | public | void |  | Инициализирует якорь ребра |
| moveDistanceLabel | private | void |  | Перерисовывает объект длины |

**Класс Node**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| dijkstraDistance | private | double | Вещественное расстояние в алгоритме Дейкстры | |
| dijkstraTexTokens | private | List<String> | Список символьных токенов в алгоритме Дейкстры | |
| visited | private | boolean | Метка посещенности для DFS | |
| processed | private | boolean | Метка обработки для алгоритма поиска циклов | |
| num | private | ArrayList<Edge> | Номер вершины | |
| RADIUS | private | double | Радиус узла | |
| color | private | Color | Базовый цвет ребра | |
| selectedColor | private | Color | Цвет ребра при выделении | |
| curColor | private | Color | Текущий цвет ребра | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| select | public | void |  | Выбирает узел |
| deselect | public | void |  | Снимает отметку выбранности |
| restore | public | void |  | Восстанавливает удаленную вершину |
| getNeighboursSorted | public | List<Node> |  | Возвращает список соседей отсортированных по номеру |
| getNeighbours | public | Set<Node> |  | Возвращает список соседей |
| getNeighboursAndCOunts | public | Map<Node, Integer> |  | Возввращает словарь соседей и количество ребер между соседом и вершиной |
| rescaleX | public | void | Double scale | Меняет положение узла по оси X при изменении размера области отрисовки |
| rescaleY | public | void | Double scale | Меняет положение узла по оси Y при изменении размера области отрисовки |
| addEdge | public | boolean | Edge edge | Добавляет ребро |
| removeNeighbour | public | void | Node n | Удаляет вершину из соседей |
| moveCenter | private | void | Double x, double y | Перемещает центр узла |
| create | public | boolean |  | Создает узел |
| unvisit | public | void |  | Помечает узел как непосещенный (и все связанные ребра) |
| handleEdges | private | void | Consumer<Edge> handler | Применяет функцию к каждому ребру из вершины |
| reconnect | private | void |  | Пересоединяет все ребра вершины |
| checkBoundsCrossed | private | boolean[] | MouseEvent event | Проверка на пересечение границы области отрисовки |

**Класс Graph**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| nodes | private | ArrayList<Node> | Вершины графа | |
| MAX\_SIZE | private | int | Максимальное количество вершин | |
| instance | private | Graph | Синглтон | |
| showDistances | private | boolean | Показываются ли расстояния | |
| selectedNode | private | Node | Выбранный узел для функции путей | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| selectNode | public | void | Node n | Выбирает узел ля функции путей |
| deselectNode | public | void |  | Снимает отметку выбранности |
| areDistancesShown | public | boolean |  | Показываются ли длины ребер |
| addNode | public | void | Node node | Добавляет узел в граф |
| removeNode | public | void | Node node | Возвращает список соседей |
| refereshLabels | public | void | Node node | По удаленному узлу обновляет номера вершин |
| getEdgesAndDistances | public | HashMap<Edge, Pair<String, Double>> |  | Список всех ребер и их длин |
| getOrientedEdgesCount | public | int |  | Считает количество ориентированных ребер |
| setNew | public | void | Graph newGraph | Стирает текущий граф и задает новый с перерисовкой |
| rescale | public | void | Char axis, double oldVal, double newVal | Обновляет положение графа при изменении области отрисовки |
| hideLengths | public | void |  | Прячет длины ребер |
| resetLengths | public | void |  | Сбрасывает длины ребер в бесконечность |
| changeLengths | public | void | String input | Задает все длины равными переданному значению |

**Класс Cache**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| MAX\_COMMANDS | private | int | Максимальное число команд в стэке | |
| commandStack | private | Command[] | Стэк команд (массив с двумя указателями) | |
| currentSize | private | Int | Количество команд в массиве | |
| undoCommandPointer | private | Int | Указывает на последнюю отмененную команду | |
| redoCommandPointer | private | int | Указатель для последнюю повторенную команду | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| push | public | Command | Command command | Обновляет указатели и 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 SetSingleLengthCommand**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| length | private | EdgeDistance | Изменяемая длина |
| oldLengthText | private | String | Старая текстовая длина |
| oldRealValue | private | Double | Старая вещественная длина |
| newLengthText | private | String | Новая текстовая длина |
| newRealValue | private | Double | Новая вещественная длина |

**Класс CreateCommand**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| created | private | Undoable | Созданный элемент |

**Класс DeleteCommand**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| deleted | private | Undoable | Удаленный элемент |
| connectedEdges | private | ArrayList<Edge> | В случае есть была удалена вершина надо сохранить ее ребра |

**Класс SetAllLengthsCommand**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| newLength | private | Double | Новая вещественная длина |
| initialized | private | boolean | Были ли инициализированы длины |
| newTextLength | private | String | Новая текстовая длина |
| oldLengthValues | private | HashMap<Edge, Pair<String, Double>> | Ребра и их текстовые и вещественные длины до обновления |

**Класс InterfaceController**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| @FXML-аналоги элементов контроля в интерфейсе | | | | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| openFile | private | void |  | Открывает файловый диалог |
| saveUnchanged | private | void |  | Сохранение графа |
| discardAndOpen | private | void |  | Открывает файл и сбрасывает несохраненные данные |
| hideDialog | private | void |  | Прячет файловый диалог |
| createNode | private | void | MouseEvent event | Создание узла по клику |
| clearWorkingArea | private | void |  | Очищает область отрисовки |
| undoAction | private | void |  | Отменяет действие |
| redoAction | private | void |  | Повторяет отмененное действие |
| drawLengths | private | void |  | Отображает метки на ребрах |
| removeLengths | private | void |  | Прячет метки на ребрах |
| toInftyLengths | private | void |  | Задает длины равными бесконечности |
| setLengths | private | void |  | Задает всем ребрам одну длину |
| calculateMagnitude | private | void |  | Запускает вычисление магнитуды по графу |
| calculateIhara | private | void |  | Запускает вычисление функции Ихары на вершинах |
| calculateEdgeIhara | private | void |  | Запускает вычисление функции Ихары на ребрах |
| calculatePathIhara | private | void |  | Запускает вычисление функции Ихары на путях |
| calculateCustomFunction | private | void |  | Запускает вычисление функции на путях |

**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 DrawingAreaController**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| BOUNDS\_GAP | public | int | Минимальное расстояние от края области отрисовки до узла | |
| instance | private | DrawingAreaController | Инстанс | |
| pane | private | AnchorPane | Панель отрисовки | |
| fileDialog | private | StackPane | Файловый диалог | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| clear | public | void |  | Удаляет все элементы с поля |
| hideNode | public | void | Javafx.scene.Node | Убирает элемент с поля |
| addNode | public | void | Javafx.scene.Node nde | Отрисовывает элемент в поле |
| createNodeByClick | public | Node | MouseEvent event | Создает узел по событию клика мыши |
| drawNodeLayout | private | Node | Double xPos, double yPos | Рисует непосредственно сам узел (круг и номер внутри) |
| checkBoundsCrossed | private | Double[] | Double xPos, double yPos | Проверка на выход за границы отрисовки |

**Класс FileManager**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| chosenFile | private | File | Выбранный пользователем файл | |
| functionFileChooser | private | FileChooser | Реализует файловый диалог для записи результата вычисления функции в файл .tex/.txt | |
| fileChooser | private | FileChooser | Реализует файловый диалог для записи графа в файл .graph | |
| mainStage | private | Stage | Сцена JavaFX | |
| dontNeedSave | private | BooleanProperty | Булево поле, не нужно сохранение (проперти надо для того чтобы отслеживать изменения в графе и деактивировать кнопку когда надо) | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| save | public | void |  | Сохранение в текущий файл или новый, если такого нет |
| saveAs | public | void |  | Сохранение в выбранный файл |
| openGraphFile | public | void |  | Открывает выбранный пользователем файл (с показом файлового диалога) |
| saveFunctionOutput | public | void | String functionResult | Сохраняет результат функции в файл |
| serializeCurrentGraph | private | void | File file | Сериализует и сохраняет в файл граф |

**Класс EventFilter**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| dragging | private | boolean | Осуществляется ли dragging в текущий момент | |
| edgeStarted | private | boolean | Создается ли новое ребро в текущий момент | |
| editing | private | boolean | Изменяется ли граф в текущий момент | |
| nodeWithStartedEdge | private | Node | Узел, из которого начато ребро | |
| startedEdge | private | Edge | Начатое из вершины ребро (но не законченное) | |
| 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** |
| deleteNotEndedEdge | private | void |  | Удаляет незаконченное ребро |

**Класс Invoker**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| commands | private | Cache | Кэш комманд | |
| instance | private | Invoker | Синглтон | |
| lastSaveCommand | private | Command | Последняя команда, на которой граф сохранялся | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| reset | Package private | void |  | Очищает кэш |
| renewLastSaveCommand | Package private | void |  | Задает последнюю сохраненную команду как текущую |
| checkLastSaveCommand | Package private | void |  | Проверка на изменение графа (нужно для активации-деактивации кнопки сохранения) |
| create | public | void | Undoable el | Создает элемент |
| setAllLengths | public | void | String length | Меняет все длины |
| setSingleLength | public | void | EdgeDistance length, String textVal, double doubleVal | Меняет конкретную длину ребра |
| delete | public | void | Undoable toDelete | Удаляет элемент |
| undoCommand | public | void |  | Вызывает undo у последней команды |
| redoCommand | public | void |  | Вызывает redo у последней отмененной команды |

**Класс MenuManager**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| edgeMenu | private | EdgeContextMenu | Контекстное меню ребра |
| nodeMenu | private | NodeContextMenu | Контекстное меню вершины |

**Класс CommonContextMenu**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Методы | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Аргументы** | **Назначение** |
| show | public | | void | | | javafx.scene.Node node, double x, double y | Показывает контекстное меню |
| Поля | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Назначение** | | |
| undoable | | protected | | Undoable | Удаляемый элемент для которого открывается контекстное меню | | |

**Класс Token**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| value | private | String | Токен в польской нотации (знак операции или число) |

**Класс Operation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** | |
| operationPriority | private | int | Чем выше число, тем раньше выполняется операция | |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| execute | public | Double | Double x, double y | Применяет операцию к переданным значениям |

**Класс Function**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| rootDegree | private | | double | | | Степень извлекаемого корня | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| execute | | public | | double | Double x | | Извлекает корень из переданного аргумента |

**Класс Parser**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| mathOps | private | | HashMap<Character, Operation> | | | Поддерживаемые математические операции | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| parseMathExpressionToRealNumber | | public | | double | String input | | Считает по строковому математическому выражению вещественный результат |
| toPostfixNotation | | private | | void | ArrayDeque<Token> queue, ArrayDeque<Token> stack, String input | | Преобразовывает токены в польскую нотацию |
| texToSympy | | private | | String | List<String> tokens | | Преобразует список токенов в выражение вещественная часть + иррациональная |
| parseTexToSympy | | private | | String | Strin token | | Преобразует иррациональный токен latex в sympy |

**Класс PopupMessage**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| messageLabel | private | | Label | | | Текстовый лейбл | |
| transition | private | | FadeTransition | | | Анимация | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| setMessageLabel | | public | | void | Label label | | Инициализация |
| showPopup | | public | | void | String message | | Показывает всплывающее окно |

**Класс EdgeData**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| from | private | int | Номер вершины из которой выходит ребро |
| to | private | int | Номер вершины куда приходит ребро |
| index | private | int | Индекс ориентированного ребра |

**Класс CustomFunctionDto**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| paths | private | List<String> | Длины всех путей |
| vertex | private | int | Номер вершины |

**Класс IharaDto**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| A | private | int[][] | Матрица смежности |
| W | private | String[][] | Взвешенная матрица |
| Q | private | int[][] | Диагональная матрица |
| rm1 | private | int | |E|-|V| |
| weighted | private | boolean | Взвешен ли граф |

**Класс IharaEdgeDto**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| edgeMatrix | private | int[][] | Матрица ребер |
| edgeOrder | private | String | Словарь ориентированных ребер |

**Класс IharaPathDto**

|  |  |  |  |
| --- | --- | --- | --- |
| Поля | | | |
| **Имя** | **Доступ** | **Тип** | **Назначение** |
| edgeMatrix | private | int[][] | Матрица ребер |
| edgeOrder | private | String | Словарь ориентированных ребер |
| spanningTree | private | String | Словарь ребер в остовном дереве |
| notSpanningTree | private | String | Словарь ребер вне остовного дерева |

**Класс IharaZetaFunctionService**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| calculateIharaFunction | public | void |  | Вычисляет функцию Ихары на вершинах |
| calculateIharaEdgeFunction | public | void |  | Вычисляет функцию Ихары на ребрах |
| calculateIharaPathFunction | public | void |  | Вычисляет функцию Ихары на путях |
| constructDto | Public | IharaDto |  | Формирует данные для Ихары на вершинах |
| validate | private | void |  | Валидирует граф |

**Класс CustomFunctionService**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| calculate | public | void |  | Вычисляет функцию на путях |
| validate | private | void |  | Валидирует граф |

**Класс MagnitudeService**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Методы | | | | |
| **Имя** | **Доступ** | **Тип** | **Аргументы** | **Назначение** |
| calculate | public | void |  | Вычисляет функцию магнитуд |
| validate | private | void |  | Валидирует граф |

**Класс PythonService**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поля | | | | | | | |
| **Имя** | **Доступ** | | **Тип** | | | **Назначение** | |
| OBJECT\_MAPPER | private | | ObjectMapper | | | Маппер java-объектов в json | |
| Методы | | | | | | | |
| **Имя** | | **Доступ** | | **Тип** | **Аргументы** | | **Назначение** |
| constructResult | | public | | void | Object dto, String scriptPath, String dataPath | | Запускает скрипт и возвращает строку результата вычисления функции |
| runScript | | public | | String | String pathToScript | | Запускает скрипт по указанному пути |
| writeMatrix | | public | | void | List<String> args, String pathToData | | Записывает матрицу в файл |
| writeJsonObject | | Public | | void | Object value, String pathToData | | Записывает объект в json-файл |

**ihara.py, custom.py, ihara\_edge.py, ihara\_path.py, magnitude.py, utils.py**

**ihara.py – вычисление функции Ихары на вершинах**

|  |  |  |
| --- | --- | --- |
| Поля | | |
| **Имя** | **Тип** | **Назначение** |
| DATA\_FILE\_PATH | str | Путь к файлу с данными |
| SAMPLE\_FILE\_PATH | str | Путь к файлу с шаблоном файла-ответа |
| A\_PREFIX | str | Префикс строки для вставки матрицы смежности |
| W\_PREFIX | str | Префикс строки для вставки матрицы весов |
| Q\_PREFIX | str | Префикс строки для вставки диагональной матрицы |
| DET\_PREFIX | str | Префикс строки для вставки определителя |
| INV\_PREFIX | str | Префикс строки для вставки обратной функции |
| RES\_PREFIX | str | Префикс строки для вставки результирующей функции |
| A | arr | Матрица смежности или матрица весов |
| Q | arr | Диагональная матрица |
| rm1 | int | |E|-|V| |
| weighted | bool | Признак взвешенности графа |
| u | symbol | Символ-аргумент функции |

|  |  |  |  |
| --- | --- | --- | --- |
| Методы | | | |
| **Имя** | **Возвращает** | **Аргументы** | **Назначение** |
| read\_data |  |  | Читает данные из json файла — матрицу смежности или матрицу весов, диагональную матрицу, |E|-|V|, признак взвешенности и записывает в global переменные |
| calc\_det | Str output (обновленный результат), expr calculated(вычисленный определитель) | Output (итоговая строка), m\_A(матрица смежности в sympy), m\_Q(диагональная матрица в sympy) | Принимает матрицу смежности и диагональную матрицу и по ним вычисляет определитель I-m\_A\*u + m\_Q\*u\*u. Записывает итог в latex строку. |
| write\_result | Str output(обновленный результат) | output(итоговая строка), det (определитель с предыдущего шага) | Вычисляет выражение (1-u\*u)\*^rm1 \*det и берет от него обратное. Записывает конечный результат в строку. |
| main |  |  | Управляет флоу вычислений, принимает решение, взвешенный или не взвешенный вариант вычислять и основываясь на нем делает разный вывод |

**custom.py – вычисление функции на путях графа**

|  |  |  |
| --- | --- | --- |
| Поля | | |
| **Имя** | **Тип** | **Назначение** |
| DATA\_FILE\_PATH | str | Путь к файлу с данными |
| SAMPLE\_FILE\_PATH | str | Путь к файлу с шаблоном файла-ответа |
| VERTEX\_PREFIX | str | Префикс строки для вставки номера стартовой вершины |
| RES\_PREFIX | str | Префикс строки для вставки результирующей функции |
| t | symbol | Символ-аргумент функции |

|  |  |  |  |
| --- | --- | --- | --- |
| Методы | | | |
| **Имя** | **Возвращает** | **Аргументы** | **Назначение** |
| read\_data | Paths (все длины путей), vertex(номер стартовой вершины) |  | Читает данные из json файла — стартовую вершину и символьные пути (преобразовывая их в sympy-выражения) |
| evaluate | Итоговый результат в формате LaTeX | Paths (список sympy-выражений путей из вершины) | Вычисляет по всем путям l произведение выражений t/(1-e^(l\*t)) и записывает в реультат |
| main |  |  | Управляет флоу вычислений и печатает результат |

**ihara\_edge.py – вычисление функции Ихары на ребрах**

|  |  |  |
| --- | --- | --- |
| Поля | | |
| **Имя** | **Тип** | **Назначение** |
| DATA\_FILE\_PATH | str | Путь к файлу с данными |
| SAMPLE\_FILE\_PATH | str | Путь к файлу с шаблоном файла-ответа |
| EDGES\_PREFIX | str | Префикс строки для вставки словаря ориентированных ребер |
| RES\_PREFIX | str | Префикс строки для вставки результирующей функции |

|  |  |  |  |
| --- | --- | --- | --- |
| Методы | | | |
| **Имя** | **Возвращает** | **Аргументы** | **Назначение** |
| read\_data | W(матрица смежности ребер), edge\_order(словарь ребер) |  | Читает данные из json файла — матрицу смежности рёбер и словарь рёбер |
| form\_matrix | W\_symb(символьная матрица ребер с комплексными числами) | W(матрица смежности ребер) | Принимает матрицу смежности ребер и преобразует ее в итоговую матрицу, где стоит комплексное число если ребра смежны и 0 иначе |
| main |  |  | Записывает результат в строку и выводит ее |

**ihara\_path.py – вычисление функции Ихары на путях**

|  |  |  |
| --- | --- | --- |
| Поля | | |
| **Имя** | **Тип** | **Назначение** |
| DATA\_FILE\_PATH | str | Путь к файлу с данными |
| SAMPLE\_FILE\_PATH | str | Путь к файлу с шаблоном файла-ответа |
| EDGES\_PREFIX | str | Префикс строки для вставки словаря ориентированных ребер |
| RES\_PREFIX | str | Префикс строки для вставки результирующей функции |
| TREE\_PREFIX | str | Префикс строки для вставки ребер входящих в остовное дерево |
| NOT\_TREE\_PREFIX | str | Префикс строки для вставки ребер не входящих в остовное дерево |

|  |  |  |  |
| --- | --- | --- | --- |
| Методы | | | |
| **Имя** | **Возвращает** | **Аргументы** | **Назначение** |
| read\_data | W(матрица ребер), edge\_order(словарь ребер), spanning\_tree (строка ребер остовного дерева), not\_spanning\_tree(строка ребер вне остовного дерева ) |  | Читает данные из json файла — матрицу рёбер, словарь рёбер, ребра остовного дерева и ребра вне дерева |
| main |  |  | Записывает результат в строку и выводит ее |

**magnitude.py – вычисление магнитуды графа**

|  |  |  |
| --- | --- | --- |
| Поля | | |
| **Имя** | **Тип** | **Назначение** |
| DATA\_FILE\_PATH | str | Путь к файлу с данными |
| SAMPLE\_FILE\_PATH | str | Путь к файлу с шаблоном файла-ответа |
| DELIMITER | str | Разделитель |
| REPLACE\_Z | str | Префикс строки для вставки матрицы кратчайших путей |
| REPLACE\_INV\_Z | str | Префикс строки для вставки обратной матрицы кратчайших путей |
| REPLACE\_SUM | str | Префикс строки для вставки суммы (итоговое выражение) |
| matrix | arr | Матрица кратчайших путей |
| output | str | Результирующая строка |
| COMPLEX | bool | Есть ли иррациональные числа в матрице |
| q | symbol | Символ-аргумент функции |

|  |  |  |  |
| --- | --- | --- | --- |
| Методы | | | |
| **Имя** | **Возвращает** | **Аргументы** | **Назначение** |
| insert\_matrix | Sympy матрица Z |  | По матрице кратчайших путей формирует матрицу Z |
| insert\_inverted\_matrix | Inverted (инфертированная матрица Z) | Матрица Z | Вычисляет обратную матрицу Z, выбирая стратегию исходя из того, есть ли в ней иррациональность |
| insert\_sum | S (сумма элементов инвертированной матрицы Z) | z\_inv(инвертированная матрица с предыдущего шага) | Суммирует все элементы переданной матрицы |
| parse\_args |  |  | Читает из файла с данными матрицу путей и признак иррациональности и записывает в глобальные переменные |
| read\_sample |  |  | Считывает шаблон в глобальную строку output |
| main |  |  | Управляет флоу вычислений, выводит итоговый результат |

**utils.py – вспомогательные функции**

|  |  |  |
| --- | --- | --- |
| Поля | | |
| **Имя** | **Тип** | **Назначение** |
| BEGIN\_MATRIX | str | Строка-начало матрицы в LaTeX |
| END\_MATRIX | str | Строка-конец матрицы в LaTeX |
| MATRIX\_DELIMITER | str | Разделитель элементов в матрице |
| q | symbol | Символ-аргумент функции |

|  |  |  |  |
| --- | --- | --- | --- |
| Методы | | | |
| **Имя** | **Возвращает** | **Аргументы** | **Назначение** |
| read\_sample | Str output | Str file\_path | Считывает шаблон в строку |
| convert\_matrix\_row\_to\_string | Ряд матрицы в формате LaTeX | Row (ряд матрицы) | Каждый элемент переводится в LaTeX формат и формируется строка матрицы |
| write\_matrix | Matrix (latex-атрица) | Prefix, input\_matrix | Возвращает latex-представление матрицы |
| write\_matrix\_and\_replace | Output(результирующая строка), sympy-матрица из исходной | Prefix, output, input\_matrix | Возвращает latex-представление матрицы и записывает ее в ответ |

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

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