|  |  |
| --- | --- |
| ***Подп. и дата*** |  |
| ***Инв. № дубл.*** |  |
| ***Взам. инв. №*** |  |
| ***Подп. и дата*** |  |
| ***Инв. № подл*** | RU.17701729.04.13-01 81 01-1-ЛУ |

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Доцент департамент больших данных и информационного поиска факультета компьютерных наук, к.ф.-м.н.    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. Л. Чернышев «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г. | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Шилов «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

**ПРОГРАММА ДЛЯ НАХОЖДЕНИЯ ВРЕМЕНИ НАСЫЩЕНИЯ ДЛЯ ОРИЕНТИРОВАННЫХ МЕТРИЧЕСКИХ ГРАФОВ**

**Пояснительная записка**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.04.13-01 81 01-1-ЛУ**

Исполнитель  
студент группы БПИ196  
\_\_\_\_\_\_\_\_\_\_ / Татаринов Н.А. /  
«\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г.

**Москва 2020**

УТВЕРЖДЕНRU.17701729.04.13-01 81 01-1-ЛУ

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл | RU.17701729.04.13-01 81 01-1 |

**Пояснительная записка**

**RU.17701729.04.13-01 81 01-1**

**Листов 29**

**Москва 2020**

**АННОТАЦИЯ**

В данном программном документе приведена пояснительная записка к программе «Программа для нахождения времени насыщения для ориентированных метрических графов» (“Program for Finding Saturation Time for Directed Metric Graphs”).

В разделе «Введение» указано наименование программы, краткое наименование программы и документы, на основании которых ведется разработка.

В разделе «Назначение и область применения» указано функциональное назначение программы, эксплуатационное назначение программы и краткая характеристика области применения программы.

В разделе «Технические характеристики» содержатся следующие подразделы:

* постановка задачи на разработку программы;
* описание алгоритма и функционирования программы с обоснованием выбора схемы алгоритма решения задачи и возможные взаимодействия программы с другими программами;
* описание и обоснование выбора метода организации входных и выходных данных;
* описание и обоснование выбора состава технических и программных средств.

В разделе «Ожидаемые технико-экономические показатели» указана предполагаемая потребность и экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами

Настоящий документ разработан в соответствии с требованиями:

1. ГОСТ 19.101-77 Виды программ и программных документов [1];
2. ГОСТ 19.102-77 Стадии разработки[2];
3. ГОСТ 19.103-77 Обозначения программ и программных документов[3];
4. ГОСТ 19.104-78 Основные надписи[4];
5. ГОСТ 19.105-78 Общие требования к программным документам[5];
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом[6];
7. ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению [7].

Изменения к Пояснительной записке оформляются согласно ГОСТ 19.603-78 [8], ГОСТ 19.604-78 [9].

**Содержание**

[**1. ВВЕДЕНИЕ** 5](#_Toc41005191)

[**1.1.** **Наименование программы** 5](#_Toc41005192)

[**1.2.** **Краткая характеристика области применения** 5](#_Toc41005193)

[**1.3 Наименование темы разработки** 5](#_Toc41005194)

[**2. НАЗНАЧЕНИЕ РАЗРАБОТКИ** 6](#_Toc41005195)

[**2.1. Функциональное назначение** 6](#_Toc41005196)

[**2.2. Эксплуатационное назначение** 6](#_Toc41005197)

[**3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ** 7](#_Toc41005198)

[**3.1.** **Постановка задачи на разработку программы** 7](#_Toc41005199)

[**3.2.** **Описание алгоритма и функционирования программы** 7](#_Toc41005200)

[**3.2.1.** **Добавление/смещение вершин и рёбер.** 8](#_Toc41005201)

[**3.2.2.** **Проверка полученного графа на корректность.** 9](#_Toc41005202)

[**3.2.3.** **Моделирование распространения эпсилон-окрестностей по графу** 9](#_Toc41005203)

[**3.3.** **Описание и обоснование выбора метода организации входных и выходных данных** 9](#_Toc41005204)

[**3.4.** **Описание и обоснование выбора состава технических и программных средств** 10](#_Toc41005205)

[**4. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ** 11](#_Toc41005206)

[**4.1. Предполагаемая потребность** 11](#_Toc41005207)

[**4.2. Ориентировочная экономическая эффективность** 11](#_Toc41005208)

[**4.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами** 11](#_Toc41005209)

[**ПРИЛОЖЕНИЕ 1 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ** 13](#_Toc41005210)

[**ПРИЛОЖЕНИЕ 2 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ** 14](#_Toc41005211)

[**ПРИЛОЖЕНИЕ 3 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ** 15](#_Toc41005212)

[**ПРИЛОЖЕНИЕ 4 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ** 17](#_Toc41005213)

[**ПРИЛОЖЕНИЕ 5 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ** 18](#_Toc41005214)

[**ПРИЛОЖЕНИЕ 6 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ** 19](#_Toc41005215)

[**ПРИЛОЖЕНИЕ 7 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ** 21](#_Toc41005216)

[**ПРИЛОЖЕНИЕ 8 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ** 22](#_Toc41005217)

[**ПРИЛОЖЕНИЕ 9 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ** 24](#_Toc41005218)

[**ПРИЛОЖЕНИЕ 10 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ** 25](#_Toc41005219)

[**ПРИЛОЖЕНИЕ 11 ДИАГРАММА КЛАССОВ** 27](#_Toc41005220)

# **1. ВВЕДЕНИЕ**

* 1. **Наименование программы**

Наименование программы – «Программа для нахождения времени насыщения для ориентированных метрических графов» (“Program for Finding Saturation Time for Directed Metric Graphs”).

* 1. **Краткая характеристика области применения**

«Программа для нахождения времени насыщения для ориентированных метрических графов» – программа, реализующая алгоритм построения и визуализации сильно-связных ориентированных метрических графов, моделирующая распространение эпсилон-окрестностей по этим графам и позволяющая пользователю визуально найти время насыщение с некоторым приближением.

Задача программы заключается в обеспечении возможности построить и визуализировать сильно-связный ориентированный метрический граф, построить модель распространения эпсилон-окрестностей по этому графу и предоставить возможность по этой модели определить приблизительное значение времени насыщения.

Программа может использоваться студентами и преподавателями при исследовании динамических систем на метрических графах.

**Документы, на основании которых ведется разработка**

Приказ Национального исследовательского университета "Высшая школа экономики"

№ 2.3-02/1112-04 от 11.12.2019.

**1.3 Наименование темы разработки**

Наименование темы разработки – «Разработка программы для нахождения времени насыщения для ориентированных метрических графов» (“Development of a Program for Finding Saturation Time for Directed Metric Graphs”).

Программа выполняется в рамках темы курсовой работы в соответствии с учебным планом подготовки бакалавров по направлению 09.03.04 «Программная инженерия» Национального исследовательского университета «Высшая школа экономики», факультет компьютерных наук, департамент программной инженерии.

# **2. НАЗНАЧЕНИЕ РАЗРАБОТКИ**

## **2.1. Функциональное назначение**

Программа будет применяться для построения и визуализации сильно-связных ориентированных метрических графов, а также для нахождения приблизительного времени насыщения для этих графов (в учебных и научных целях).

## **2.2. Эксплуатационное назначение**

Программа будет использоваться для нахождения и анализа ориентировочного значения времени насыщения, полученного в ходе работы с заданным графом. Программный продукт позволит решать некоторые задачи, возникающие при исследовании динамических систем на метрических графах.

# **3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ**

## **Постановка задачи на разработку программы**

Программа должна обеспечивать возможность выполнение перечисленных ниже функций:

1. Задание сильно-связного ориентированного метрического графа с помощью графического интерфейса и сохранение этого графа в файл.
2. Задание сильно-связного ориентированного метрического графа из файла.
3. Анализ входных данных на корректность.
4. Моделирование распространения эспилон-окрестностей по рёбрам заданного графа.
5. Предоставление возможности визуального приблизительного определения времени насыщения для заданного графа.

## **Описание алгоритма и функционирования программы**

Хранение графа в программе разделяется на 2 формы: математическая форма (классический список смежности, с которым можно производить различные манипуляции, к которому возможно применять различные алгоритмы, к примеру, поиск количества компонент сильной связности) и графическая форма [10] (элементы, позволяющие визуализировать граф на экране).

Для математической формы была разработана библиотека классов GraphLib, состоящая из двух классов. Первый – обобщённый класс Pair<T, U>, содержащий в себе два свойства – T First и U Second (первый и второй элемент в паре соответственно) и методы/операторы сравнения двух пар. Для решения поставленной перед нами задачи мы используем тип int в качестве T и тип double в качестве U, причём первый элемент пары – номер конечной вершины на ребре графа (целое число, большее 0), а второй элемент пары – вес ребра (действительное число, большее либо равное 1 и меньшее либо равное 100). Второй класс – статический MetricOriented, содержащий статические методы для работы с графом (считывание с файла и запись в него, добавление нового ребра и другие). В основной программе математическая форма графа представляется в виде списка смежности List<Pair<int, double>>[] graph, где graph[i][j] – ребро, идущее из вершины i в вершину graph[i][j].First и имеющее вес graph[i][j].Second.

Для графической формы были разработаны 2 класса, лежащие в том же пространстве имён, что окна WPF. Первый класс – Vertex, представляющий вершину графа. Задаётся координатами центра, X и Y, кольцом Ring (представляется типом Ellipse с одинаковыми высотой и шириной, с толстыми стенками) и находящимся внутри него блоком текста RingContent (представляется типом TextBox с неизменяемым свойством Text, содержащим номер вершины). В основной программе для выведения вершин графа на экран используется массив графических вершин Vertex[] drawnVertices, причём от каждой вершины на экран выводится Ring и RingContent. Второй класс – Edge, представляющий ребро графа. В первую очередь, задаётся тремя точками Point First, Point Middle и Point Final – начальная, промежуточная и конечная точки соответственно. Точки First и Final могут менять своё значения в процессе выполнения программы (во время перемещения вершин), а точка Middle вычисляется автоматически: для этого используется специальное число Number, получаемое единожды в конструкторе от пользователя, и две переменные смещения (приватные поля) deltaX и deltaY, значения которых зависят от расположения точек First и Final. Для пересчёта deltaX и deltaY в зависимости от First и Final есть метод CalculateDelta(). Number, передаваемый в качестве параметра в конструкторе, вычисляется как количество рёбер между First и Final до добавления данного ребра. Также, в Edge присутствуют 3 элемента, которые выводятся на экран и образуют ребро. Этими элементами являются ломаная DrawnEdge (отрезок и First в Middle и из Middle в Final), текстовый блок Weight, содержащий значение веса ребра в свойстве Text (привязан левым верхним углом к точке Middle), и треугольник Arrow, отражающей направление ребра (лежит на отрезке из Middle в Final и вычисляется с помощью метода CreateArrow()).

Помимо графа в программе необходимо хранить какое-либо представление эпсилон-окрестности. Для этого создан класс EpsilonNeighbourhood. Он содержит начальную, промежуточную и конечную точки ребра, по которому движется окрестность, а также индексы этого ребра (для перехода на следующее ребро). На экран выводится ломаная Neighbourhood, за смещение которой отвечают переменные delta1X, delta1Y (по первой половине ребра), delta2X, delta2Y (по второй половине ребра). Смещение происходит в методе MoveNeighbourhood, для работы которого необходимо хранить ещё ряд переменных.

Изначально пользователю предлагается 2 варианта для получения графа: из файла или через графический интерфейс. Но важно помнить, что после чтения из файла граф попадёт на тот же самый графический интерфейс (подробнее описано в пункте 3.3).

### **Добавление/смещение вершин и рёбер.**

Для данной задачи есть 3 метода, отвечающие 3-м событиям мыши: опускание левой кнопки мыши, смещение мыши и поднятия левой кнопки мыши; и 2 кнопки на экране: если одну из них можно нажать, то вторая заблокирована, и наоборот. Копки отвечают за то, что мы сейчас добавляем: вершины или рёбра.

Если вершины, то метод при опускании левой кнопки мыши работает так: если, опустив мышь, мы попали на какую-либо вершину (то есть внутрь Ring какой-то вершины), то мы запоминаем её индекс, координаты её центра и меняем её цвет, а затем «захватываем мышь», отслеживая её координаты; если же мы не попали ни в одну вершину, то мы добавляем новую (при условии, что он находится не ближе определённого расстояния от всех остальных и не ближе определённого расстояния от границ экрана). Метод при движении мыши работает следующим образом: если мышь «захвачена», то мы перемещаем вершину по координатам курсора (при этом, мы помним начальные); если же мышь не «захвачена», то перемещение мыши ни на что не влияет. И, наконец, метод при поднятии левой кнопки мыши: если мышь была «захвачена», то мы оставляем перемещаемую вершину там, где остановился курсор при поднятии (если это возможно по условиям расстояний – если невозможно, то возвращаем к начальному положению).

Если отмечаем рёбра, то метод опускания левой кнопки мыши проверяет, попали ли мы на какую-то вершину. Если попали и индекс для запоминания равен -1 (не указывает на вершину), то запоминаем (без «захвата» мыши), выделяем отдельным цветом. Если попали и индекс не равен (-1), то проводим ребро между двумя вершинами. Если не попали, то либо ничего не происходит, либо высвобождается вершина, которую запоминали. Остальные методы ничего не выполняют при отмечании рёбер.

### **Проверка полученного графа на корректность.**

Получаем ли мы файл через граф или сразу задаём его в графическом интерфейсе – в любом случае перед моделированием будет необходимо выполнить проверку графа на корректность через графический интерфейс. Условия корректности графа:

1. В графе есть хотя бы одно ребро.
2. Веса всех рёбер (полученные из текстовых блоков на экране) должны быть действительными числами в диапазоне [1; 100].
3. Граф обязан быть сильно-связным (вычисляется через метод FindAmountOfStronglyConnectedComponents(List<Pair<int, double>> graph) – подробнее в приложении).

Кроме того, для запуска моделирования необходимо, чтобы были заданы величина эпсилон-окрестности (действительно число в диапазоне (0; 0.5)), номер стартовой вершины (целое число от 1 до количества вершин в графе включиельно), путь к файлу, в который сохранется итоговый файл.

### **Моделирование распространения эпсилон-окрестностей по графу**

У каждого экземпляра класса EpsilonNeighbourhood есть метод MoveNeighbourhood(), смещающий окрестность на шаг, заданный заранее. В окне моделирования запускается таймер, во время каждого тика которого в отдельном потоке [11], [12] запускается метод, смещающий все существующие на момент этого тика эпсилон-окрестности на шаг. При этом перед этими вычислениями таймер отключается, чтобы следующий тик таймера не вызывался раньше, чем окончатся вычисления метода смещения (иначе возникнет исключение).

## **Описание и обоснование выбора метода организации входных и выходных данных**

Пользователь имеет возможность задать входной граф двумя способами. Первый – через графический интерфейс, в котором пользователь собственноручно отмечает на экране вершины и рёбра (имея возможность их перемещать). Второй способ – с помочью файла, имеющего следующий формат (являющийся списком смежности):

1) В первой строке должно находиться целое положительное число, не превосходящее максимального значения типа int, - количество вершин в графе.

2) Во второй строке должно находиться целое положительное число, не превосходящее максимального значения типа int, - количество рёбер в графе.

3) Далее должно следовать столько строк, сколько рёбер есть в графе (число из второй строки). В каждой из этих строк должны быть ровно 3 числа, разделённые произвольным количеством пробелов: первое число – номер (от 1 до количества вершин в графе включительно) начальной вершины ребра; второе число – номер (от 1 до количества вершин в графе включительно) конечной вершины ребра; третье число – вес ребра (от 1 до 100 включительно – не имеет смысла производить моделирование на графе, у которого веса рёбер отличаются более, чем на 2 порядка).

При чтении графа из файла производится проверка на описанные выше условия, однако не производится проверка на сильную связность графа. Далее, данный список смежности отправляется в графический интерфейс, являющийся альтернативным способом для задания графа, где вершины распределяются по экрану случайным образом, рёбра присоединяются к ним по специальному алгоритму (учитывающему наличие произвольного количества кратных рёбер). Данная реализация выбрана для того, чтобы при считывании графа из файла всегда была возможность передвинуть/добавить вершины, добавить новые рёбра, изменить веса старых рёбер. Если же пользователю необходимо продолжить выполнение без изменений, он может нажать кнопку перехода к моделированию без каких-либо изменений. При этом, при нажатии этой кнопки вне зависимости от того, как до этого был задан граф, производится проверка на сильную связность и сохранение графа в файл, введённый пользователем, если он полностью корректен. После всех манипуляций с вводом/изменением графа запускается моделирование.

## **Описание и обоснование выбора состава технических и программных средств**

Для нормального функционирования программы требуется компьютер, оснащенный следующими техническими компонентами:

1. процессор не ниже Intel Core i7-5500U CPU или совместимый с ним с тактовой частотой не ниже 2.4 ГГц;
2. 16 Гб ОЗУ или более;
3. монитор с разрешением не ниже 1920х1080;
4. клавиатура и мышь.

Для нормального функционирования программы требуется компьютер, оснащенный следующими программными компонентами:

1. операционная система Microsoft Windows 10;
2. библиотека Microsoft .NET Framework 4.8 и выше;
3. среда программирования – Microsoft Visual Studio 2019 и выше.

# **4. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ**

## **4.1. Предполагаемая потребность**

Программа будет использоваться студентами и преподавателями для исследования динамических систем на метрических графах.

## **4.2. Ориентировочная экономическая эффективность**

Программа бесплатно предоставляет определённый функционал для работ в прикладной математике и информатике в области теории графов.

## **4.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами**

Программа будет бесплатной и будет иметь русскоязычный интерфейс.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1) ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

2) ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

3) ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

4) ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

5) ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

7) ГОСТ 19.404-79 Поснительная записка. Требования к содержанию и оформлению, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

8) ГОСТ 19.603-78 Общие правила внесений изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

9) ГОСТ 19.604-78 Правила внесений изменений в программные документы, выполненные печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

10) Руководство по WPF [Электронный ресурс]// URL: <https://metanit.com/sharp/wpf/> (режим доступа: свободный, дата обращения: апрель 2020)

11) Оператор await (Справочник по C#) [Электронный ресурс]// URL <https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/await> (режим доступа: свободный, дата обращения: апрель 2020)

12) async (Справочник по C#) [Электронный ресурс]// URL <https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/async> (режим доступа: свободный, дата обращения: апрель 2020)

# **ПРИЛОЖЕНИЕ 1 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ**

Таблица 1.1

**Описание и функциональное назначение классов библиотеки классов GraphLib**

|  |  |
| --- | --- |
| **Класс** | **Назначение** |
| MetricOrientedGraph | Статический класс, содержащий методы для работы с ориентированными метрическими графами. |
| Pair<T,U> | Обобщенный класс, представляющий пару, первый элемент которой является типом T, а второй – типом U. |

Таблица 1.2

**Описание и функциональное назначение классов проекта SimulationWPF**

|  |  |
| --- | --- |
| **Класс** | **Назначение** |
| MainWindow | Является стартовым окном, предоставляющим выбор способа получения графа (из файла или через графический интерфейс). |
| FileGraph | Окно, предоставляющее возможность считать граф из файла. При успешном считывании передает полученный граф в графический интерфейс. |
| DrawnGraph | Окно, предоставляющее пользователю возможность собственноручно задать граф и перейти в окно моделирования |
| Simulation | Окно, моделирующее распространение эпсилон-окрестностй по переданному в него графу. |
| Edge | Класс, представляющий нарисованное ребро ориентированного метрического графа. |
| Vertex | Класс, представляющий нарисованную вершину ориентированного метрического графа. |
| EpsilonNeighbourhood | Класс, представляющий нарисованную эпсилон-окрестность ориентированного метрического графа. |

# **ПРИЛОЖЕНИЕ 2 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ**

Таблица 2.1

**Описание статических методов класса MetricOrientedGraph**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Модификатор доступа** | **Тип возвращаемого значения** | **Аргументы** | **Назначение** |
| SaveGraphToFile | public | void | string path, List<Pair<int, double>>[] graph | Записывает граф graph в файл path. |
| ReadGraphFromFile | public | List<Pair<int, double>>[] | string path | Считывает граф из файла path. |
| AddEdgeToGraph | public | int | List<Pair<int, double>>[] graph,  int vertex, Pair<int, double> edge | Добавляет ребро edge с начальной вершиной vertex в граф. |
| FindAmountOfStronglyConnectedComponents | public | int | List<Pair<int, double>>[] graph | Вычисляет количество компонент сильной связности в графе. |
| DFS | public | void | List<Pair<int, double>>[] graph, int vertex, bool[] visited | Поиск в глубину по графу из вершины с номеров vert, определющий, какие из вершин были посещены. |
| DFS | public | void | List<Pair<int, double>>[] graph, int vertex, bool[] visited, List<int> order | Поиск в глубину по графу из вершины с номеров vert, определющий, какие из вершин были посещены и в каком порядке. |
| GetTransposeGraph | public | List<Pair<int, double>>[] | List<Pair<int, double>>[] graph | Возвращает обратный (транспонированный) переданному граф. |

# **ПРИЛОЖЕНИЕ 3 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ**

Таблица 3.1

**Описание полей класса Pair<T,U>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Модификатор доступа** | **Тип** | **Назначение** |
| first | private | T | Первый элемент пары. |
| second | private | U | Второй элемент пары. |

Таблица 3.2

**Описание свойств класса Pair<T,U>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Модификатор доступа** | **Тип** | **Назначение** |
| First | public | T | Первый элемент пары. При попытке присвоения значения производит проверку того, что оно не является null. |
| Second | public | U | Второй элемент пары. При попытке присвоения значения производит проверку того, что оно не является null. |

Таблица 3.3

**Описание методов класса Pair<T,U>**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Модификатор доступа** | **Тип возвращаемого значения** | **Аргументы** | **Назначение** |
| Pair | public | Конструктор | **-** | Присваивает элементам пары значения по умолчанию. |
| Pair | public | Конструктор | T \_first, U \_second | Присваивает элементам пары переданные значения. |
| Equals | public | bool | object obj | Переопределенный метод, проверяющий, равен ли переданный объект текущему. |
| Equals | public | bool | Pair<T, U> other | Метод, проверяющий, равен ли переданный объект текущему. |
| GetHashCode | public | int | - | Переопределенный метод, служит хэш-функцией для объекта. |
| ToString | public | string | - | Представляет текущий объект |
| CompareTo | public | int | Pair<T, U> other | Сравнивает текущий объект с переданным по первому элементу, а в случае равенства первых – по второму. |

Таблица 3.4

**Описание перегруженных операторов класса Pair<T,U>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Модификатор доступа** | **Тип** | **Назначение** |
| > | public | bool | Возвращает true, если первый объект больше второго. |
| < | public | bool | Возвращает true, если первый объект меньше второго. |
| >= | public | bool | Возвращает true, если первый объект больше или равен второму. |
| <= | public | bool | Возвращает true, если первый объект меньше или равен второму. |
| == | public | bool | Возвращает true, если первый объект равен второму. |
| != | public | bool | Возвращает true, если первый объект не равен второму. |

# **ПРИЛОЖЕНИЕ 4 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ**

Таблица 4.1

**Описание методов класса MainWindow**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Модификатор доступа** | **Тип возвращаемого значения** | **Аргументы** | **Назначение** |
| MainWindow | public | Конструктор | **-** | Происходит инициализация компонентов. |
| ExitProgram\_Click | private | void | object sender, RoutedEventArgs e | Выход из программы. |
| FileChoice\_Click | private | void | object sender, RoutedEventArgs e | Переход в форму для считывания графа из файла. |
| DrawChoice\_Click | private | void | object sender, RoutedEventArgs e | Переход в форму рисования графа. |

# **ПРИЛОЖЕНИЕ 5 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ**

Таблица 5.1

**Описание методов класса FileGraph**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Модификатор доступа** | **Тип возвращаемого значения** | **Аргументы** | **Назначение** |
| FileGraph | public | Конструктор | **-** | Происходит инициализация компонентов. |
| GetGraph\_Click | private | void | object sender, RoutedEventArgs e | Попытка считывания графа из файла для перехода в форму рисования. |
| ExitProgram\_Click | private | void | object sender, RoutedEventArgs e | Выход из прграммы. |

# **ПРИЛОЖЕНИЕ 6 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ**

Таблица 6.1

**Описание полей класса DrawnGraph**

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Модификатор доступа** | **Тип** | **Назначение** |
| amOfVertices | private | int | Количество вершин в графе. |
| amOfEdges | private | int | Количество ребер в графе. |
| graph | private | List<Pair<int, double>>[] | Метрический ориентированный граф. |
| drawnVertices | private | Vertex[] | Массив графических вершин. |
| drawnEdges | private | List<Edge>[] | Массив списков графических ребер. |
| index | private | int | Индекс вершины для запоминания. |
| startX | private | int | X-координата центра захваченной вершины. |
| startY | private | int | Y-координата центра захваченной вершины. |

Таблица 6.2

**Описание методов класса DrawnGraph**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Модификатор доступа** | **Тип возвращаемого значения** | **Аргументы** | **Назначение** |
| DrawnGraph | public | Конструктор | **-** | Инициализирует массивы пустыми элементами. |
| DrawnGraph | public | Конструктор | List<Pair<int, double>>[] \_graph | Инициализирует граф подготовленным заранее графом. |
| ExitProgram\_Click | private | void | object sender, RoutedEventArgs e | Выход из программы. |
| DrawVertices\_Click | private | void | object sender, RoutedEventArgs e | Переход на режим отмечания и перемещения вершин. |
| DrawEdges\_Click | private | void | object sender, RoutedEventArgs e | Переход на режим отмечания рёбер. |
| FindSaturationTime\_Click | private | void | object sender, RoutedEventArgs e | Проиводит попытку получения графа, сохранения его в файл и запуска формы моделирования. |
| PlaceToDraw\_MouseLeftButtonDown | private | void | object sender, MouseButtonEventArgs e | Опускание левой кнопки мыши. |
| PlaceToDraw\_MouseMove | private | void | object sender, MouseEventArgs e | Перемещение мыши. |
| PlaceToDraw\_MouseLeftButtonUp | private | void | object sender, MouseButtonEventArgs e | Поднятие левой кнопки мыши. |

# **ПРИЛОЖЕНИЕ 7 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ**

Таблица 7.1

**Описание полей класса Simulation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Модификатор доступа** | **Тип** | **Назначение** |
| graph | private | List<Pair<int, double>>[] | Метрический неориентированный граф. |
| drawnVertices | private | Vertex[] | Массив графических вершин. |
| drawnEdges | private | List<Edge>[] | Массив списков графических ребер. |
| neighs | private | List<EpsilonNeighbourhood> | Список эпсилон-окрестностей. |
| epsilon | private | double | Величина эпсилон-окрестности. |
| tmr | private | DispatcherTimer | Таймер для смещения. |
| timeForUser | private | double | Время, выводимое пользователю. |

Таблица 7.2

**Описание методов класса Simulation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Модификатор доступа** | **Тип возвращаемого значения** | **Аргументы** | **Назначение** |
| Simulation | public | Конструктор | List<Pair<int, double>>[] \_graph, Vertex[] \_drawnVertices, List<Edge>[] \_drawnEdges, double \_epsilon, int \_startVertex | Инициализирует поля входными параметрами. |
| SetSimulationBtn\_Click | private | void | object sender, RoutedEventArgs e | Останавливает или возобновляет моделирование. |
| async TmrTick | private | void | object sender, EventArgs e | Происходит при каждом тике таймера, вызывает смещение эпсилон-окрестностей. |
| MoveAllNeighs | private | List<EpsilonNeighbourhood> | - | Смещает все эпсилон-окрестности на шаг. |
| ExitProgram\_Click | private | void | object sender, RoutedEventArgs e | Выход из программы |

# **ПРИЛОЖЕНИЕ 8 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ**

Таблица 8.1

**Описание полей класса Edge**

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Модификатор доступа** | **Тип** | **Назначение** |
| first | private | Point | Центр начальной вершины ребра. |
| final | private | Point | Центр конечной вершины ребра. |
| deltaX | private | double | Смещение точки между начальной и средней точкой ребра по X. |
| deltaY | private | double | Смещение точки между начальной и средней точкой ребра по Y. |

Таблица 8.2

**Описание свойств класса Edge**

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Модификатор доступа** | **Тип** | **Назначение** |
| First | public | Point | Центр начальной вершины ребра. При попытке присвоения значения производит пересчет характеристик ребра. |
| Middle | public | Point | Промежуточная вершина ребра. Определяется начальной и конечной вершинами. |
| Final | public | Point | Центр конечной вершины ребра. При попытке присвоения значения производит пересчет характеристик ребра. |
| Number | public | int | Специальное число, задающее расположение промежуточной точки. |
| DrawnEdge | public | Polyline | Ребро графа. |
| Arrow | public | Polygon | Стрелка на ребре. |

Таблица 8.3

**Описание методов класса Edge**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Модификатор доступа** | **Тип возвращаемого значения** | **Аргументы** | **Назначение** |
| Edge | public | Конструктор | Point \_first, Point \_last, int \_number | Создает ребро графа по начальной точке, конечной и специальному числу. |
| CalculateDelta | private | void | - | Вычисляет переменные смещения для промежуточной точки. |
| CreateArrow | private | void | - | Создаёт стрелку направления для ребра. |

# **ПРИЛОЖЕНИЕ 9 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ**

Таблица 9.1

**Описание полей класса Vertex**

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Модификатор доступа** | **Тип** | **Назначение** |
| x | private | int | X-координата центра вершины. |
| y | private | int | Y-координата центра вершины. |

Таблица 9.2

**Описание свойств класса Vertex**

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Модификатор доступа** | **Тип** | **Назначение** |
| X | public | int | X-координата центра вершины. |
| Y | public | int | Y-координата центра вершины. |
| Ring | public | Ellipse | Кольцо, обозначающее границы вершины. |
| RingContent | public | TextBox | Номер вершины, расположенный внутри неё. |

Таблица 9.3

**Описание методов класса Vertex**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Модификатор доступа** | **Тип возвращаемого значения** | **Аргументы** | **Назначение** |
| Vertex | public | Конструктор | int \_x, int \_y, int \_ringContent | Инициализирует поля переданными значениями. |

# **ПРИЛОЖЕНИЕ 10 ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ И СВОЙСТВ**

Таблица 10.1

**Описание полей класса EpsilonNeighbourhood**

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Модификатор доступа** | **Тип** | **Назначение** |
| backPosition | private | int | Расположение задней точки окрестности. |
| frontPosition | private | int | Расположение передней точки окрестности. |
| drawnEdgeLength | private | double | Длина ребра на экране. |
| epsilon | private | double | Длина эпсилон-окрестности на экране. |
| coef | private | double | Коэффициент отношения линейных значений на экране к номинальным линейным значениям. |
| delta1X | private | double | Смещение по оси OX на первой части ребра при каждом шаге. |
| delta1Y | private | double | Смещение по оси OY на первой части ребра при каждом шаге. |
| delta2X | private | double | Смещение по оси OX на второй части ребра при каждом шаге. |
| delta2Y | private | double | Смещение по оси OY на второй части ребра при каждом шаге. |

Таблица 10.2

**Описание свойств класса EpsilonNeighbourhood**

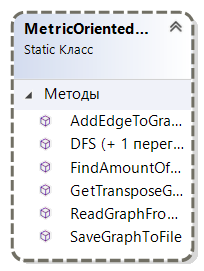
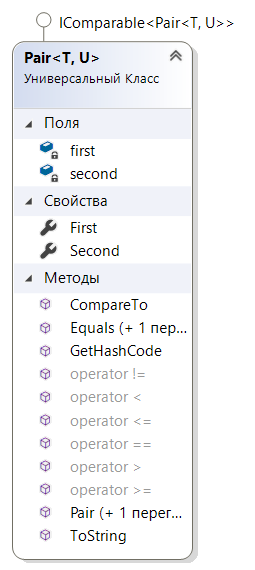
|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Модификатор доступа** | **Тип** | **Назначение** |
| FirstPoint | public | Point | Начальная точка ребра, по которому передвигается окрестность. |
| MiddlePoint | public | Point | Промежуточная точка ребра, по которому передвигается окрестность. |
| FinalPoint | public | Point | Конечная точка ребра, по которому передвигается окрестность. |
| EdgeIndexI | public | int | i-й индекс ребра, по которому передвигается окрестность. |
| EdgeIndexJ | public | int | j-й индекс ребра, по которому передвигается окрестность. |
| Neighbourhood | public | Polyline | Передвигающаяся по экрану епсилон-окрестность. |

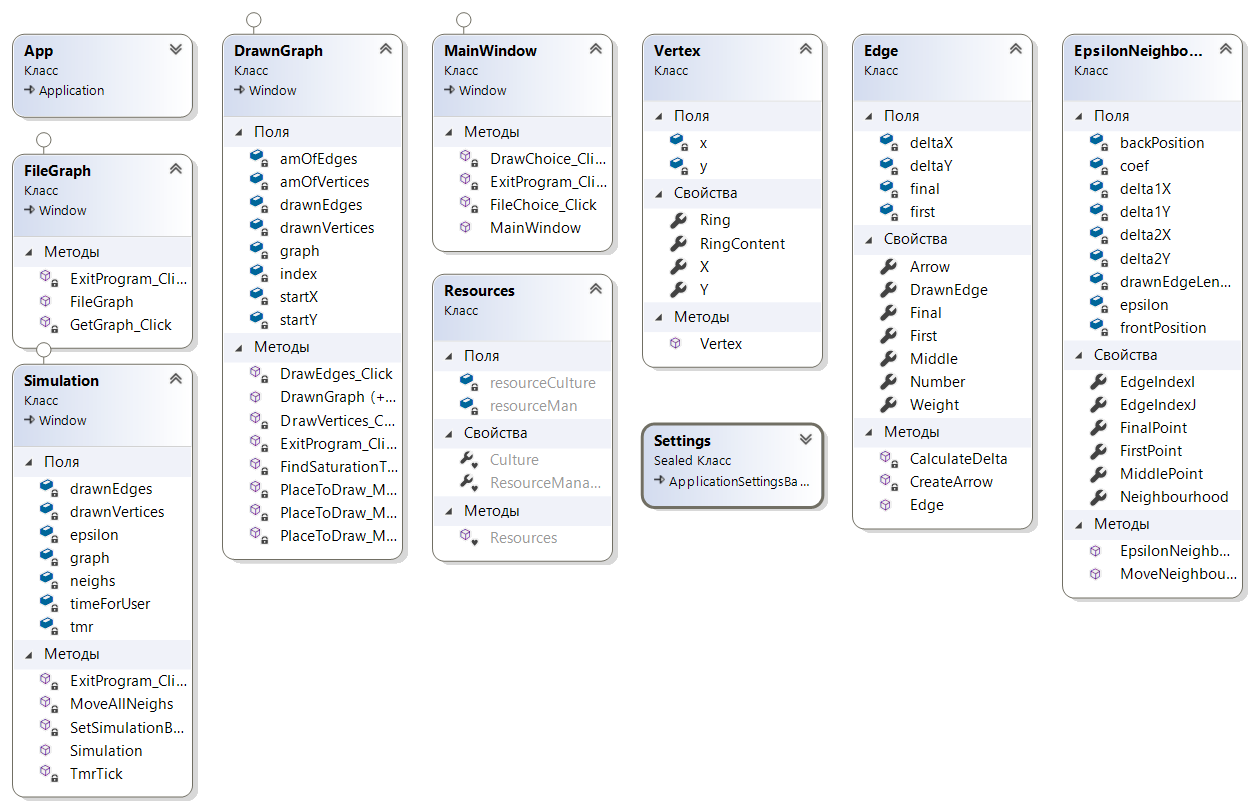
Таблица 10.3

**Описание методов класса** **EpsilonNeighbourhood**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Модификатор доступа** | **Тип возвращаемого значения** | **Аргументы** | **Назначение** |
| EpsilonNeighbourhood | public | Конструктор | Point \_firstPoint,  Point \_middlePoint, Point \_finalPoint,  int \_edgeIndexI, int \_edgeIndexJ,  double \_edgeWeight, double \_startPosition,  double \_epsilon, double \_step | Инициализирует экземпляр эпсилон-окрестности через параметры. |
| MoveNeighbourhood | public | double | - | Метод, пересчитывающий положение эпсилон-окрестности при смещении на шаг. |

# **ПРИЛОЖЕНИЕ 11 ДИАГРАММА КЛАССОВ**





|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Изм. | Номера листов | | | | Всего листов (страниц) в документе | № документа | Входя- щий № сопро- водит. докум. и дата | Под- пись | Дата |
| изменен- ных | заменен- ных | новых | аннулиро- ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |