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

Факультет компьютерных наук

Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Кандидат технических наук, доцент департамента программной инженерии факультета компьютерных наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К. Ю. Дегтярев  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2019 г. |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. Инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл.*** | RU.17701729.04.0 9-01 ПЗ 01-1-ЛУ | | **Программа оценки стоимости разработки ПО с использованием нечетких деревьев решений**  **Пояснительная записка**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.501430-01 ПЗ 01-1-ЛУ**  Исполнитель  студент группы БПИ 173  \_\_\_\_\_\_\_\_\_\_\_/Королев Д. П./  «\_\_\_»\_\_\_\_\_\_\_\_\_\_2019 г.  **2019** | | |  |

УТВЕРЖДЕНО

RU.17701729.04.09-01 ПЗ 01-1-ЛУ

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. Инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл.*** | RU.17701729.04.0 9-01 ПЗ 01-1 | | **Программа оценки стоимости разработки ПО с использованием**  **нечетких деревьев решений**  Пояснительная записка  RU.17701729.04.09-01 ПЗ 01-1  **Листов 25**  **2019** |  |

**СОДЕРЖАНИЕ**

[1. Введение 4](#_Toc8350256)

[1.1. Наименование программы 4](#_Toc8350257)

[1.2. Документы, на основании которых ведется разработка 4](#_Toc8350258)

[2. Назначение и область применения 5](#_Toc8350259)

[2.1. Функциональное назначение 5](#_Toc8350260)

[2.2. Эксплуатационное назначение 6](#_Toc8350261)

[2.3. Краткая характеристика области применения 6](#_Toc8350262)

[3. Технические характеристики 7](#_Toc8350263)

[3.1. Постановка задачи на разработку программы 7](#_Toc8350264)

[3.2. Описание алгоритма и функционирования программы 7](#_Toc8350265)

[3.2.1 Библиотека LiveCharts 10](#_Toc8350266)

[3.2.2 Библиотека Shields.GraphViz 11](#_Toc8350267)

[3.3. Описание и обоснование выбора метода организации входных и выходных данных 11](#_Toc8350268)

[3.4. Описание и обоснование выбора состава технических и программных средств 11](#_Toc8350269)

[3.4.1. Описание выбора состава технических средств 11](#_Toc8350270)

[3.4.2. Обоснование выбора состава технических средств 12](#_Toc8350271)

[3.4.3. Описание выбора состава программных средств 12](#_Toc8350272)

[3.4.4. Обоснование выбора состава программных средств 12](#_Toc8350273)

[4. Ожидаемые технико-экономические показатели 13](#_Toc8350274)

[4.1. Предполагаемая востребованность 13](#_Toc8350275)

[4.2. Экономические преимущества разработки по сравнению с аналогами 13](#_Toc8350276)

[5. Источники, использованные при разработке 14](#_Toc8350277)

[6. Приложения 16](#_Toc8350278)

[6.1. Описание и функциональное назначение классов 16](#_Toc8350279)

[6.2. Описание и функциональное назначение полей, методов и свойств 17](#_Toc8350280)

# 1. Введение

## 1.1. Наименование программы

Наименование программы – «Программа оценки стоимости разработки ПО с использованием нечетких деревьев решений».

## 1.2. Документы, на основании которых ведется разработка

Основанием для разработки является приказ декана факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики» № 2.3-02/1012-02 от 10.12.18 «Об утверждении тем, руководителей курсовых работ студентов образовательной программы Программная инженерия факультета компьютерных наук».

Разработка программы проводится в рамках выполнения курсовой работы по теме «Программа оценки стоимости разработки ПО с использованием нечетких деревьев решений»

# 2. Назначение и область применения

## 2.1. Функциональное назначение

Назначение программы – получение оценки стоимости разработки программного обеспечения на основе формируемого нечеткого дерева решений с использованием задаваемых исходных данных. Исходные данные состоят из двух частей, а именно: множество (база) примеров и построенные пользователем значения (в виде нечетких множеств) выбранных лингвистических переменных.

Примеры представляются в табличной форме в виде пар типа «критерий - значение». Каждый критерий, в данном случае, является одной из многих оценивающих стоимость разработки программного обеспечения характеристик – например, это может быть «размер команды разработчиков, шт. человек» или «опыт команды разработчиков, кол-во лет»). Считается, что количество критериев варьируется и не превосходит десяти. Анализ критериев и их выбор основан на публикациях [8,9] – в дальнейшем, список используемых источников может быть расширен.

## 2.2. Эксплуатационное назначение

Программа предназначена для пользователей, ставящих перед собой задачу получения предварительных (ориентировочных) оценок стоимости разработки программного обеспечения. Программа ориентирована на менеджеров проектов, стейкхолдеров проектов, разработчиков и студентов (с целью обучения и знакомства с рассматриваемым подходом).

## 2.3. Краткая характеристика области применения

Решение данной задачи имеет потенциальное приложение в разных областях:

1. Предварительная оценка стоимости разработки программного обеспечения менеджером или стейкхолдером некоторого проекта.
2. Аналитическое сравнение методологий оценки стоимости программного обеспечения с алгоритмом, использующим нечеткое дерево решений.

# 3. Технические характеристики

## 3.1. Постановка задачи на разработку программы

Функции, указанные ниже, должны быть реализованы в программе:

1. Настройка множества значений лингвистических переменных. Возможность редактирования путем задания (ввода) параметров и их графического отображения.
2. Ввод числовых и словесных значений критериев рассматриваемого ПО для создания лингвистической переменной.
3. Формирование нечеткого дерева решений для вычислений оценки стоимости разработки ПО.
4. Получение входных числовых значений каждой из лексикографических переменных.
5. Получение результата в виде численной оценки стоимости разработки ПО, а также графического представления нечеткого дерева решений, в соответствии с которым данная оценка была получена.

## 3.2. Описание алгоритма и функционирования программы

Алгоритм, описанный в статье «A Fuzzy Decision Tree Algorithm Based on C4.5» [1], является основным алгоритмом, примененным в программе.

Лингвистическая переменная [2] – переменная *x*  U , значениями которой являются термы, выраженные словами или словосочетаниями на естественном языке (в словесной форме). Каждый терм *A* ассоциирован с нечетким множеством, представляемым функцией принадлежности μ *A* : U  [0,1] . В частности, такие функции могут использоваться в нечетких деревьях решений (например, критерий «размер команды разработчиков, шт. человек» может быть описан, в простом случае, множеством значений – {«малый», «средний», «большой»}); соответствие значения критерия словесной интерпретации не всегда является четким, т.е. в большинстве случаев имеет «размытые» границы. Например, «размер команды разработчиков, шт. человек» может быть с точки зрения восприятия и оценки специалистами «малым» на прибл. 80% и «средним» на прибл. 20%.

Т-норма [3] – ассоциативная и коммутативная бинарная операция на L = [0,1]. В нечеткой логике применяется в качестве операций конъюнкции и дизъюнкции. В данной работе в качестве основной используемой т-нормы была выбрана операция минимума, поэтому коэффициент принадлежности примера к атрибуту будет высчитываться как: , где – степень принадлежности примера к узлу , - множество всех примеров узла .

Процесс построения нечеткого дерева решений начинается с вычисления степени принадлежности каждого из примеров к лингвистическим значениям (далее – ЛЗ). На рисунке 1 приведен пример-макет полученной таблицы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № примера | Атрибут 1 | | | Атрибут 2 | | |
| ЛЗ1 1 | ЛЗ1 2 | ЛЗ1 3 | ЛЗ2 1 | ЛЗ2 2 | ЛЗ3 3 |
| 1 | 0 | 1 | 0 | 0,6 | 0,4 | 0 |
| 2 | 0,2 | 0,2 | 0,6 | 0,1 | 0,5 | 0,4 |

*Рисунок 1*

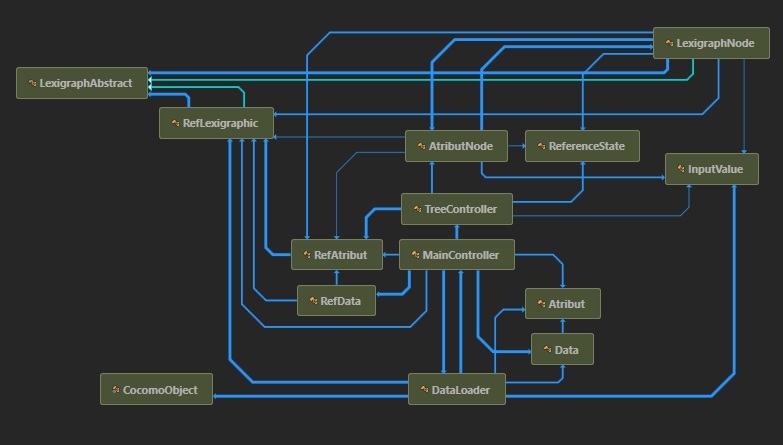
Для каждого атрибута сумма значений принадлежности к лингвистическим значениям должна быть равна единице по свойствам т-нормы.

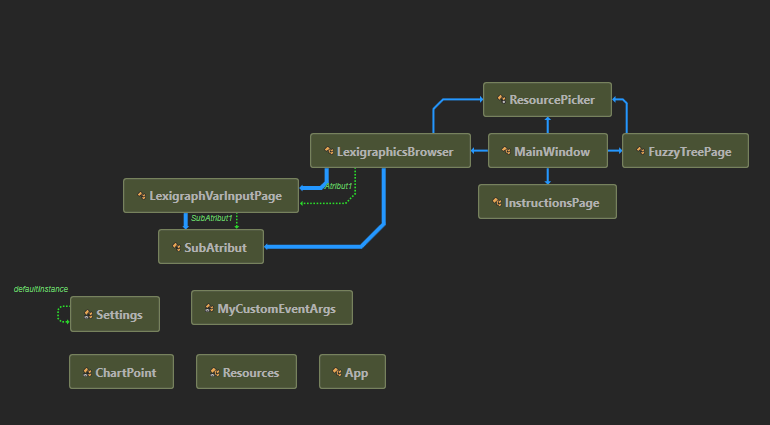
По полученной таблице рассчитываются общие параметры для проведения дальнейших вычислений – оценка среднего количества информации для определения класса объекта из множества по формуле , где - множество объектов.

Наконец, рассчитывается энтропия для разбиения по атрибуту со значениями : . Получив значения энтропии по каждому атрибуту происходит расчет значения функции прироста информации для определения класса объекта из множества: . Атрибуты сортируются в порядке возрастания функций прироста информации. Именно в этом порядке атрибуты и будут представлены в нечетком дереве решений.

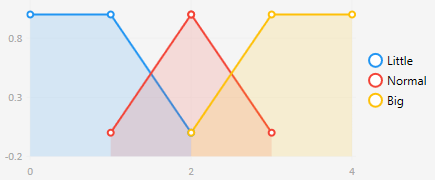
Остается лишь проверить принадлежность входных данных к целевому классу: , где – коэффициент соотношения примеров листа дерева для значения целевого класса , – степень принадлежности примера узлу , – принадлежность значения целевого класса положительному значению исхода классификации.

Программа в свою очередь состоит из 3-х проектов: библиотека, написанная на C# для реализации вышеизложенного алгоритма, консольное приложение для тестирования библиотеки (состоит из единственного класса, который использовался для запуска тестов, поэтому ниже не приведена диаграмма классов данного проекта) и приложение с графическим интерфейсом, являющееся конечным поставляемым продуктом. Ниже представлены диаграммы зависимостей классов в проектах (рис. 2 – библиотека, рис. 3 – оконное приложение).

  
*Рисунок 2*

  
*Рисунок 3*

### 3.2.1 Библиотека LiveCharts



*Рисунок 4*

Для отображения графиков (рис 4) использовалась открытая библиотека LiveCharts, использующаяся для .NET-проектов [5].

### 3.2.2 Библиотека Shields.GraphViz

Для отрисовки построенного нечеткого дерева решений и его представления в графическом виде использовалась библиотека Shields.GraphViz [6]. Предварительно в папку с проектом была установлено данное ПО, чтобы библиотека могла им воспользоваться.

## 3.3. Описание и обоснование выбора метода организации входных и выходных данных

Формат входных данных ограничен программой. Изначально программа предустанавливает входные параметры по умолчанию. При невозможности сопоставления значения лингвистической переменной с заданной кусочно-линейной функцией от данного параметра пользователю должно выдаваться окно с соответствующим сообщением об ошибке в входных параметрах.

Формат выходных данных – выданные сообщения об успешном вычислении и получаемые результаты.

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

### 3.4.1. Описание выбора состава технических средств

Минимальные и рекомендуемые свойства компьютера:

1. Процессор Intel Core i3 с частотой 2000 МГц или более быстрый (рекомендуется не менее 3000 МГц) с подходящей материнской платой.
2. Не менее 1 ГБ оперативной памяти (рекомендуется не менее 4 ГБ).
3. Не менее 1 ГБ свободного места на жестком диске.
4. Периферия для персонального компьютера, а также клавиатура.
5. Видеокарта и монитор, способные воспроизводить графическое отображение работы программы с разрешением не менее 800х600 точек [5].

### 3.4.2. Обоснование выбора состава технических средств

Выбор состава технических средств обусловлен использованием ОС Microsoft Windows X и новее, а также комфортными условиями для быстродействия работы программы.

### 3.4.3. Описание выбора состава программных средств

Для запуска и работы программы требуется компьютер, со следующим предустановленным программным обеспечением:

1. операционная система Microsoft Windows 7 и новее;
2. установленный Microsoft .NET Framework 2.0.

### 3.4.4. Обоснование выбора состава программных средств

Версия Microsoft .NET Framework 4.0 выбрана как рекомендуемая версия из предлагаемых средой Microsoft Visual Studio.

Microsoft Windows 7 выбрана как последняя из поддерживаемых ОС компанией Microsoft.

# 4. Ожидаемые технико-экономические показатели

## 4.1. Предполагаемая востребованность

Программа может применяться для коммерческого и некоммерческого использования в рамках получения информации о стоимости разработки ПО стейкхолдером или менеджером проекта или исследователем соответственно.

## 4.2. Экономические преимущества разработки по сравнению с аналогами

В сети Интернет не было найдено аналогичной программы, которая предоставляет графический интерфейс для оценки стоимости ПО с использованием нечеткого дерева решений.

У данного приложения есть следующие преимущества:

1. Эксплуатация программы не требует затрат денежных средств (распространяется бесплатно).
2. Программа не подразумевает обслуживания.

# 5. Источники, использованные при разработке

1. Marcos Evandro C. A Fuzzy Decision Tree Algorithm Based on C4.5 / Marcos Evandro C. Maria C Monard, Heloise A Camargo // Mathware & Soft Computing Magazine. Vol. 20 n. 1, January 2015 г.

2. Лингвистические переменные. [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/Лингвистические переменные, свободный (дата обращения 22.11.18)

3. Т-норма или Т-конорма. [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/T-норма\_и\_t-конорма, свободный (дата обращения 09.05.19)

4. Единая система программной документации – М.: ИПК Издательство стандартов, 2000.

5. C# LiveCharts [Электронный ресурс] / Режим доступа: https://lvcharts.net/, свободный (дата обращения 09.05.19)

6. GraphViz [Электронный ресурс] / Режим доступа: https://www.graphviz.org/, свободный (дата обращения 09.05.19)

7. Системные требования для операционных систем Windows ХР [Электронный ресурс] / Режим доступа: https://support.microsoft.com/ru-ru/kb/314865, свободный (дата обращения 20.02.15)

8. Safa Mohammed A. S. Factors that influence software project cost and schedule estimation / Safa Mohammed A. S., Gada Kadoba // 2017 Sudan Conference on Computer Science and Information Technology (SCCSIT), 20 October 2017.

9. Mohammed Vth / Evaluating software cost estimation models using fuzzy decision trees / Sanaa E., Ali I. // 2018 IEEE International Conference on Software Quality, Reliability and Security Companion, 20 July 2018.

# 6. Приложения

## 6.1. Описание и функциональное назначение классов

Таблица 1. Название классов и их функциональное назначение (описание).

|  |  |
| --- | --- |
| Класс | Назначение |
| LexigraphAbstract | Абстрактный класс для лексикографического значения. |
| MainController | Класс, имеющий открытое API любых требуемых функций для реализации алгоритма. Связывает остальные классы для корректной работы приложения. Желательно использование класса в качестве singleton. |
| CocomoObject | Класс-реализация объекта из выбранной базы данных. |
| Atribut | Класс-реализация атрибута. |
| Data | Класс-контроллер для хранения всех входных параметров. |
| DataLoader | Класс-загрузчик данных из файла (базы данных) |
| InputValue | Класс, реализующий сущность входного значения. |
| RefAtribut | Класс-реализация атрибута (лексикографической переменной), внутри которого находится список объектов лексикографических значений |
| RefData | Класс-контроллер для лексикографических переменных |
| ReferenceState | Класс-реализация значения энтропии |
| RefLexigraphic | Класс-реализация лексикографического значения. Унаследован от абстрактного класса LexigraphAbstract. |
| AtributNode | Класс-реализация вершины-атрибута в дереве решений |
| LexigraphNode | Класс-реализация вершины-лексикографического значения в дереве решений. Унаследован от абстрактного класса. |
| TreeController | Контроллер для построения дерева решений, а также его отрисовки |

## 6.2. Описание и функциональное назначение полей, методов и свойств

Таблица 2. Описание и функциональное назначение полей класса LexigraphAbstract

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| Name | string | protected | Имя лексикографического значения |
| from | double | protected | Значение, с которого функция равна единице |
| to | double | protected | Значение, до которого функция равна единице |
| left | double | protected | Значение, с которого функция начинает расти с нуля |
| right | double | protected | Значение, до которого функция продолжает существовать |
| PositiveRef | double | protected | Положительное значение функции энтропии атрибута |
| NegativeRef | double | protected | Негативное значение функции энтропии атрибута |

Таблица 3. Описание и функциональное назначение методов класса LexigraphAbstract

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификаторы | Тип возвращаемого значения | Аргументы | Назначение |
| CountReference | public | double | double value | Возвращает значение функции принадлежности |

Таблица 4. Описание и функциональное назначение полей класса MainController

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| Rendered | bool | public | Флаг, указывающий на то, была ли сгенерирована картинка по построенному дереву |
| Data | Data | public | Singleton объект контроллер |
| RefData | RefData | public | Singleton объект контроллер |
| DataLoader | DataLoader | public | Singleton объект контроллер |
| TreeController | TreeController | public | Singleton объект контроллер |

Таблица 5. Описание и функциональное назначение методов класса MainController

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификаторы | Тип возвращаемого значения | Аргументы | Назначение |
| MainController | public | void | int numberOfDecimals, bool dataReinitialize = true | Конструктор для инициализации контроллеров, а так же пересоздании объекта базы входных параметров, если нужно |
| CountTillEnd | public | double | – | Используется для запуска алгоритма с начала до конца |
| LoadAtributData\_and\_RefAtributData | public | void | string name, List<double> valsDoubles, List<RefLexigraphic> lexigraphics | Загружает в объекты входные данные |
| CountReferenceValuesFromDataValues | public | void | – | Создает объекты из списков входных параметров |
| CountEntropyForAllLexigraphics | public | void | – | Рассчитывает энтропию для всех лексикографических переменных |
| NormalizeResultValues | public | void | – | Нормализирует результирующий вектор значений относительно максимального значения |
| GetDoubleResultValue | public | double | – | Строит дерево и возвращает результат вычислений |

Таблица 6. Описание и функциональное назначение полей класса CocomoObject

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| TeamExp | int | public | Опыт команды (годы) |
| ManagerExp | int | public | Опыт менеджера (годы) |
| Length | int | public | Длительность проекта (месяцы) |
| Entities | int | public | Количество сущностей в базе данных |
| Effort | int | public | Количество требуемых человеко-часов |

Таблица 7. Описание и функциональное назначение полей класса Atribut

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| Name | string | public | Имя лексикографической переменной |
| Values | List<double> | public | Список входных параметров под этот атрибут |

Таблица 8. Описание и функциональное назначение методов класса Atribut

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификаторы | Тип возвращаемого значения | Аргументы | Назначение |
| NormalizeValues | public | void | – | Нормализует значения в этом атрибуте |
| LoadDoubleList | public | void | List<double> vals | Загружает список входных значений в список этого класса |

Таблица 9. Описание и функциональное назначение полей класса Data

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| DataTable | List<Atribut> | public | Список объектов-атрибутов |
| MaxResultDouble | double | public | Отдельное поле для хранения максимального значения из списка входных значений атрибута |

Таблица 10. Описание и функциональное назначение методов класса Data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификаторы | Тип возвращаемого значения | Аргументы | Назначение |
| SaveMaxResultDouble | public | void | List<double> results | Сохраняет максимальное значение коллекции в соответствующее поле |
| GetResultDoubles | public | List<double> | – | Возвращает список чисел с плавающей точкой, являющийся вектором результирующих значений |
| NormalizeResultDoubles | public | void | – | Нормализирует значения в результирующем векторе |

Таблица 11. Описание и функциональное назначение полей класса DataLoader

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| \_Path | string | const private | Хранит значение пути до файла с базой данных примеров |
| InputValues | List<InputValue> | public | Хранит список входных значений, которые задаются в виде пар «ключ-значение» |

Таблица 12. Описание и функциональное назначение методов класса DataLoader

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификаторы | Тип возвращаемого значения | Аргументы | Назначение |
| AddInputValueAndGetItFromConsole | public | void | String name | Получает входное значение из консоли и загружает в оперативную память |
| AddInputFromGUI | public | void | string name, double value | Получает входное значение из графического интерфейса |
| GetCocomosFromCSV | private | List<CocomoObject> | – | Возвращает список объектов из базы данных из выбранного файла |
| LoadCocomoDataSet | public | void | – | Загружает все объекты из базы данных в контроллеры |

Таблица 13. Описание и функциональное назначение полей класса InputValue

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| Name | string | public | Наименование входного значения |
| Value | double | public | Числовое значение |

Таблица 14. Описание и функциональное назначение полей класса RefAtribut

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| Name | string | public | Наименование атрибута |
| Atribut | List<RefLexigraphic> | public | Список объектов лексикографических значений данного атрибута |

Таблица 15. Описание и функциональное назначение методов класса RefAtribut

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификаторы | Тип возвращаемого значения | Аргументы | Назначение |
| InfoGrowth | public | double | – | Возвращает значение функции прироста информации для заданного атрибута |
| AddRefLexigraphic | public | void | RefLexigraphic refLexigraphic | Добавляет лексикографическое значение к атрибуту |

Таблица 16. Описание и функциональное назначение полей класса RefData

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| RefTable | List<RefAtribut> | public | Список атрибутов |
| Entropy | double | public | Значение общей энтропии |

Таблица 17. Описание и функциональное назначение методов класса RefData

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификаторы | Тип возвращаемого значения | Аргументы | Назначение |
| LoadRefAtribut | public | void | string name, List<RefLexigraphic> lexigraphics | Загружает новый атрибут с заданными лексикографическими значениями в список атрибутов |
| GetEntrFuncRes | public | double | double pos, double neg | Возвращает значение общей энтропии |

Таблица 18. Описание и функциональное назначение полей класса ReferenceState

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| PosRef | double | public | Значение положительной составляющей энтропии |
| NegRef | double | public | Значение отрицательной составляющей энтропии |
| SharedRef | double | public | Общее значение энтропии |

Таблица 19. Описание и функциональное назначение полей класса RefLexigraphic

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| Values | List<double> | public | Список значений, проверяемых на данном лексиграфическом значении |

Таблица 20. Описание и функциональное назначение методов класса RefAtribut

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификаторы | Тип возвращаемого значения | Аргументы | Назначение |
| GetReferenceFunctionPoints | public | Double[] | – | Возвращает массив значений лексикографического значения |
| getEntrFuncRes | public | double | – | Возвращает значение функции энтропии |
| CountEntropyForAtribut | public | double | – | Возвращает значение функции энтропии для атрибута |

Таблица 21. Описание и функциональное назначение полей класса AtributNode

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| Name | string | public | Имя вершины в дереве |
| ChildrenLexigraphs | List<LexigraphNode> | public | Список вершин-детей – лексикографических значений |
| ParentLexigraph | LexigraphNode | public | Родительская вершина в дереве |

Таблица 22. Описание и функциональное назначение методов класса AtributNode

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификаторы | Тип возвращаемого значения | Аргументы | Назначение |
| PushInputThroughTree | public | void | List<InputValue> inputValues, List<ReferenceState> references | Высчитывает принадлежность к данной ветке дерева и пропускает вычисление ниже по дереву |
| GetEdgeStatements | public | void | HashSet<KeyValuePair<string, string>> edges | Возвращает список ребер в девере |

Таблица 23. Описание и функциональное назначение полей класса LexigraphNode

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| Ref | double | public | Значение степени принадлежности к данной вершине |
| ParentAtribut | AtributNode | public | Вершина-родитель |
| ChildAtribut | AtributNode | public | Вершина-ребенок |

Таблица 24. Описание и функциональное назначение методов класса LexigraphNode

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификаторы | Тип возвращаемого значения | Аргументы | Назначение |
| PushInputThroughTree | public | void | List<InputValue> inputValues, List<ReferenceState> references | Высчитывает принадлежность к данной ветке дерева и пропускает вычисление ниже по дереву |
| GetEdgeStatements | public | void | HashSet<KeyValuePair<string, string>> edges | Возвращает список ребер в девере |

Таблица 25. Описание и функциональное назначение полей класса LexigraphNode

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификаторы | Назначение |
| IsConstructed | bool | public | Флаг, определяющий было ли построено дерево |
| Root | AtributNode | public | Вершина-корень дерева |
| References | List<ReferenceState> | public | Список объектов для вычисления энтропий каждого ребра и вершины |

Таблица 26. Описание и функциональное назначение методов класса LexigraphNode

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Модификаторы | Тип возвращаемого значения | Аргументы | Назначение |
| ConstructTree | public | void | List<RefAtribut> refAtributs, bool toSort | Сконструировать дерево |
| GetEdgeStatements | public | void | HashSet<KeyValuePair<string, string>> edges | Возвращает список ребер в девере |
| resultFunction | public | double | List<ReferenceState> refs | Возвращает результат вычислений по всему дереву |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |