Оглавление

[Введение 5](#_Toc200378046)

[Часть 1. Теоретическая часть 7](#_Toc200378047)

[Метод анализа иерархий 7](#_Toc200378048)

[Примеры программной реализации метода анализа иерархии 12](#_Toc200378049)

[Онлайн-калькулятор МАИ 12](#_Toc200378050)

[Создание иерархической модели в MS Excel 14](#_Toc200378051)

[Модуль МАИ внутри программы Variant 16](#_Toc200378052)

[Язык C++ и фреймворк Qt 19](#_Toc200378053)

[Часть 2. Программная реализация 21](#_Toc200378054)

[Архитектура программы 21](#_Toc200378055)

[Организация хранения данных 25](#_Toc200378056)

[Интерфейс программы 27](#_Toc200378057)

[Часть 3. Пример использования 32](#_Toc200378058)

[Постановка задачи 32](#_Toc200378059)

[Пример решения 32](#_Toc200378060)

[Сравнение с другими способами решения задачи 32](#_Toc200378061)

[Возможные улучшения 32](#_Toc200378062)

[Приложение A Файл MAI.h Заголовочный файл метода анализа иерархий 33](#_Toc200378063)

[Приложение Б Файл MAI.cpp Исполняемый файл метода анализа иерархий 35](#_Toc200378064)

[Приложение В Пример оформления JSON файла 42](#_Toc200378065)

# Введение

В процессе работы необходимо зачастую выбирать из нескольких вариантов наиболее подходящий или оптимальный. К примеру, выбор лучшего проектного решения многоквартирного дома или выбор используемого набора технологий для создания программного обеспечения. В большинстве случаев выбор производят при помощи интуитивного метода, не прибегая к более продвинутым методам.

В некоторых случаях интуитивный метод не подходит, так как число критериев оценивания, может быть, большое количество и их комбинации могут быть достаточно сложными. Так критерии могут быть взаимоисключающие. К примеру, качество материала и его цена. Нам необходим наиболее качественные материалы для сооружения, но при этом бюджет проекта должен быть минимальным.

В таких случаях нам необходимо использовать метод анализа иерархий (МАИ), разработанный профессором Пенсивальского и Питтсбургского университетов Томасом Саати в 1970 году.

Метод анализа иерархий позволяет:

1. Разбить проблему на иерархические уровни (цель, критерии, подкритерии, альтернативы), что делает процесс принятия решений более прозрачным и логичным.
2. Учитывать качественные и количественные критерии, что позволяет работать с разными типами данных.
3. Легко модифицировать метод под конкретную задачу. Добавлять или исключать критериев и альтернатив, учитывать весовые коэффициенты, применять в динамических условиях

Но помимо всего прочего важен не только метод, при помощи которого мы производим выбор, но и те инструменты, благодаря которым используем нам метод.

Инструмент должен быть понятен пользователю, чтобы программой было пользоваться легко и без дополнительной подготовки (интуитивно понятным). При этом программа быть легко изменяемая, масштабируемая и работать на разных платформах.

Поэтому было принято решение разработать приложение на языке C++ с использованием фреймворка Qt и реализовать метод анализа иерархий.

# Часть 1. Теоретическая часть

## Метод анализа иерархий

Прежде чем говорить о существующих реализациях и методах решения стоит в первую очередь рассмотреть следующий вопрос. Что такое метод анализа иерархий?

Метод анализа иерархий (МАИ) — это математический метод структурирования и анализа сложных решений. Разработан американским математиком Томасом Саати в 1970-х годах. Цель метода – обеспечить системный и логический подход к принятию сложных решений в условиях множества критериев и альтернатив, помогая выбрать оптимальный вариант на основе взвешенного анализа. МАИ не дает «единственного верного решения», а распределяет варианты по степени соответствия критериям с учетом их важности. Это делает процесс выбора прозрачным и обоснованным.

Метод широко применяется в управлении, экономики, инженерных областях, логистике.

Основными аспектами МАИ являются:

1. Иерархическая структура – проблема разбивается на уровни.
   1. Цель;
   2. Группы критериев и критерии;
   3. Варианты выбора

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

Рис. 1. Пример иерархической модели

1. Попарное сравнение – критерии и варианты сравниваются попарно по шкале относительной важности.
2. Расчет весовых коэффициентов – на основе матриц сравнений вычисляются приоритеты критериев и вариантов
3. Согласованность оценок – проверяется логическая непротиворечивость суждений

Основными преимуществами метода являются:

1. Структурирование проблемы. Задача разбивается на уровни, что упрощает анализ модели;
2. Учет качественных и количественных критериев. Позволяет учесть при сравнении разные типы критериев (например: удобство расположения лестницы в квартире и общую площадь квартиры)
3. Объективизация субъективных суждений. Оценки переводятся в числовые приоритеты, что уменьшает влияние интуиции на выбор.
4. Многокритериальное решение. Оценка альтернатив происходит через анализ множественного числа факторов.
5. Проверка согласованности решений. Расчет индекса согласованности исключает противоречивы суждения (например, «А важнее Б, Б важнее В, но В важнее А»).

Несмотря на существующие достоинства у метода существует ряд недостатков:

1. субъективность оценок. На оценку сильно влияет мнения эксперта и способ оценки критерия;
2. большой объем вычислений при многих критериях;
3. требуется проверка согласованности.

Метод можно разделить на несколько этапов:

1. Выделение проблемы. Определение цели.
2. Выделение основных критериев и вариантов
3. Построение иерархии
4. Построение матриц попарных сравнений критериев и вариантов по критериям
5. Анализ полученных матриц и получение результатов

Рассмотрим пример расчета матрицы попарной сравнений критериев и альтернатив.

Матрица попарного сравнения составляется следующим образом.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Критерий 1 | Критерий 2 | … | Критерий n |
| Критерий 1 | 1 |  | … |  |
| Критерий 2 |  | 1 | … |  |
| … | … | … | … | … |
| Критерий n |  |  | … | 1 |

Табл. 1. Матрица попарного сравнения критериев

– отношение критерия i к критерию j.

Матрицы попарных сравнений альтернатив составляется аналогично, что и матрица сравнений критериев. Главное отличие заключается в том, что матрица сравнения вариантов составляется для каждого критерия отдельно. То есть число матриц попарного сравнения критериев суммарно будет равно число критериев и имеет следующий вид.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий n | Вариант 1 | Вариант 2 | … | Вариант n |
| Вариант 1 | 1 |  | … |  |
| Вариант 2 |  | 1 | … |  |
| … | … | … | … | … |
| Вариант n |  |  | … | 1 |

Табл. 2. Матрица попарного сравнения вариантов по притерию n

Анализ матриц происходит по следующему алгоритму

1. Находим сумму элементов каждого столбца по формуле
2. Делим элементы матрицы на сумму соответствующего столбца
3. Получаем нормированные матрицы и следующим этапом находим среднее значение каждой строки – это «веса» и получаем весовой столбец критериев.

– вес критерия i.

*−* число критериев.

1. Повторяем для матриц сравнения вариантов и получаем матрицы веса критериев и вариантов.

|  |  |
| --- | --- |
| Критерий 1 |  |
| Критерий 2 |  |
| … | … |
| Критерий n |  |

Табл. 3. Матрица весов критериев

|  |  |
| --- | --- |
| Вариант 1 |  |
| Вариант 2 |  |
| … | … |
| Вариант i |  |

Табл. 4. Матрица весов альтернатив по критерию N

– вес варианта I по критерию N

1. Для получения итогового результата необходимо перемножить матрицу весов альтернатив на вектор весов критериев. Получаем вектор отражающее на сколько хорошо каждый вариант соответствует критериям. У лучшего варианта число наибольшее.

– вес альтернатив

с максимальным значением говорит о том, что вариант i лучше всех соответствует критериям, то есть является оптимальным.

## Примеры программной реализации метода анализа иерархии

Существует множество способов реализации метода анализа иерархии. Прежде чем перейти к нашей реализации давай те рассмотрим несколько существующих решений.

Рассмотрим следующие варианты реализации:

1. Онлайн-калькулятор МАИ.
2. Создание иерархической модели в MS Excel.
3. Модуль внутри программы Variant.

### Онлайн-калькулятор МАИ

Существует множество всевозможных вариантов онлайн-калькуляторов МАИ с различными вариантами, и пользователь может выбрать желаемый вариант на свое усмотрение.

Рассмотрим в качестве примера вариацию МАИ реализованную на следующем сайте <https://axd.semestr.ru/upr/hierarchies.php>

Решение вопроса оптимального выбора разделено на 3 шага:

Шаг 1. Определение числа уровней иерархии и числа критериев на каждом уровне. Максимальное число уровней 5. Максимальное число критериев на каждом уровне – 7.

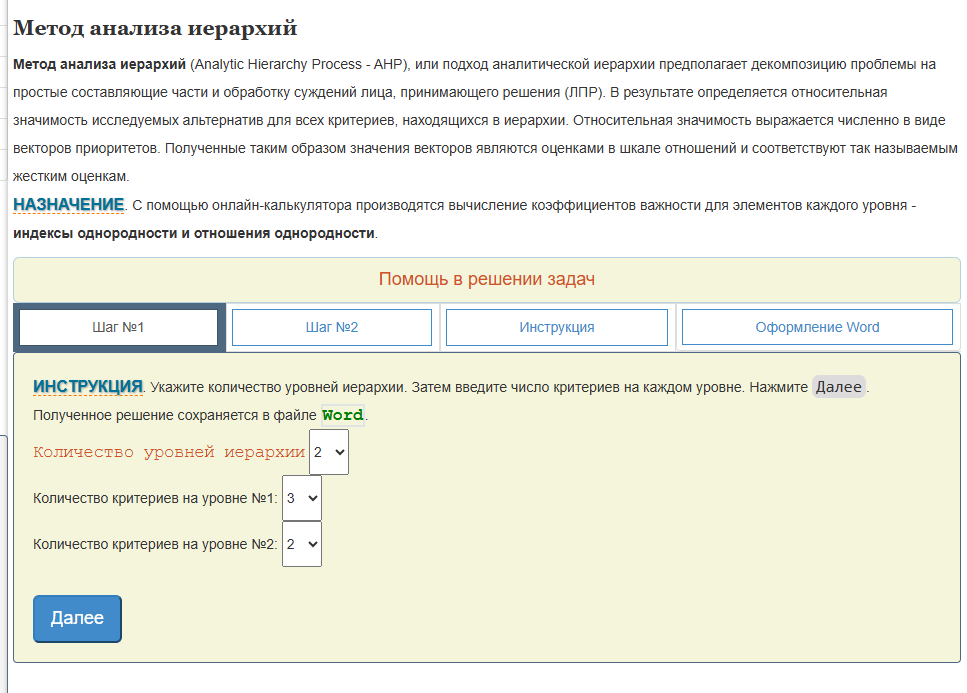


Рис. 2. Шаг 1, определение числа уровней и критериев на каждом уровне

Шаг 2. На шаге 2 пользователь задает имена критериев на каждом уровне и матрицы попарных сравнений. Пользователю в матрицах достаточно заполнить только элементы матрицы над главной диагональю, а элементы под главной диагональю рассчитываются как обратное число от противоположного элемента.

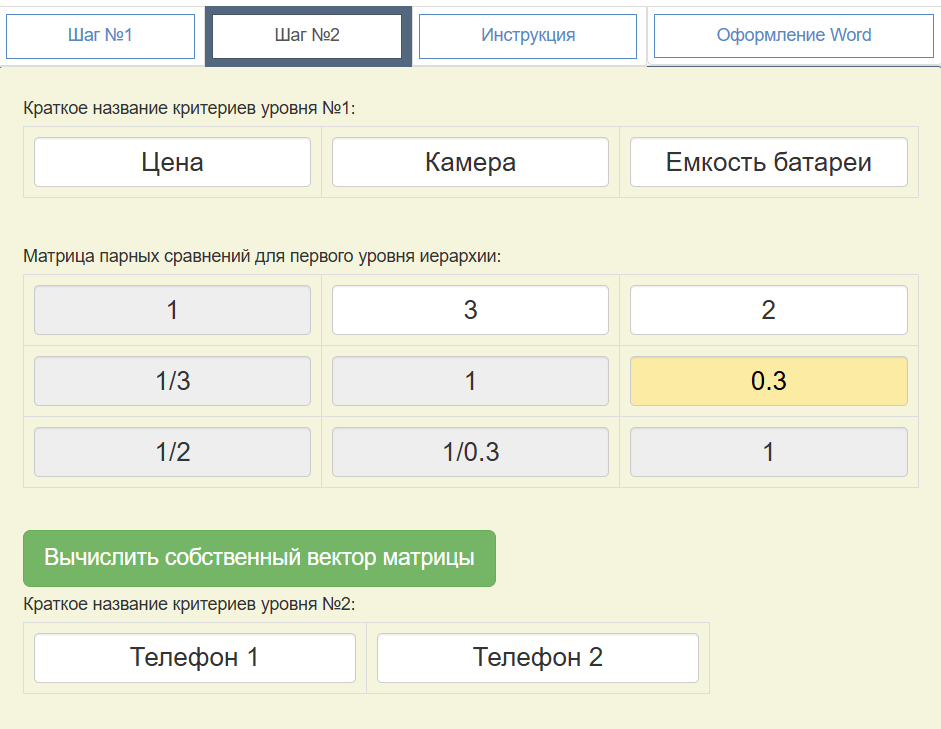


Рис. 3. Шаг 2, Заполнение имен критериев и матриц попарного сравнения (часть 1)

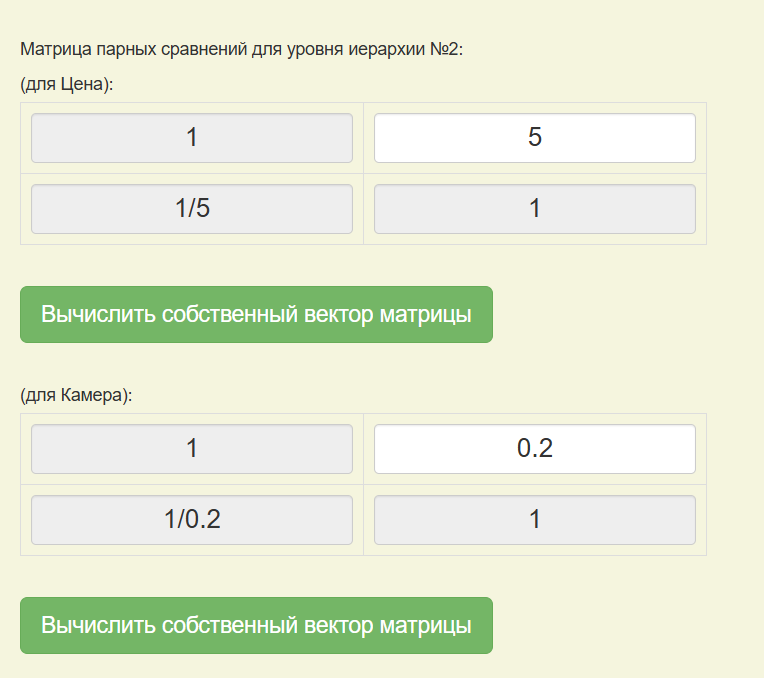


Рис. 4. Шаг 2, Заполнение имен критериев и матриц попарного сравнения (часть 2)

Изображение выглядит как текст, снимок экрана, Прямоугольник, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 5. Шаг 2, Заполнение имен критериев и матриц попарного сравнения (часть 3)

На третьем шаге (по нажатии кнопки «Далее»). Сайт выводит решение и итоговый результат. Так же есть возможность вывести решение в документ Word.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 6. Фрагмент итогового отчета

### Создание иерархической модели в MS Excel

Программа MS Excel предоставляет широкий спектр возможности по работе с таблицами и всевозможными расчетами с инструментами автоматизации (плагины и макросы VBA). Учитывая, что в МАИ необходимо работать с матрицами Excel – это идеальный вариант для решения задач такого рода.

Единственным минусом можно назвать необходимость переработки всего файла в случае изменения числа критериев и вариантов. Это значительно увеличивает время решения задачи.

Изображение выглядит как диаграмма, План, Технический чертеж, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 7. Пример реализации МАИ в MS Excel (часть 1)

Изображение выглядит как текст, электроника, снимок экрана, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 8. Пример реализации МАИ в MS Excel (часть 2)

Изображение выглядит как текст, снимок экрана, Параллельный, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 9. Пример реализации МАИ в MS Excel (часть 3)

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 10. Пример реализации МАИ в MS Excel (часть 4)

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 11. Пример реализации МАИ в MS Excel (часть 5)

Изображение выглядит как текст, линия, снимок экрана, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 12. Пример реализации МАИ в MS Excel (часть 6)

### Модуль МАИ внутри программы Variant

Программа Variant разработана кандидатом технических наук кафедры Системы автоматизированного проектирования РУТ (МИИТ) Юрием Федоровичем Тарарушкирнрым.

Предназначена решения вопросов оптимизации и разделена на два модуля:

1. Решение вопроса выбора.
2. Оптимизация конструкций методом проекции градиента

Для нас в данный момент особый интерес представляет первый модуль.

Как и прошлые реализации базируется на методе анализа иерархий. Главным отличием от предыдущих реализации заключается в том, что сравнение критериев происходит не методом попарных сравнений, что предполагает заполнение матриц, а методом визуальных сравнений, что предполагает составление рейтингов вариантом по каждому критерию. Это упрощает процесс сравнения критериев и вариантов. С другой стороны такой подход уменьшает точность сравнения критериев и альтернатив между собой.

Максимальное число уровней иерархии – 3. Это означает, что возможно поделить критерии на группы.

На следующих рисунках приведен пример решения задачи выбора.

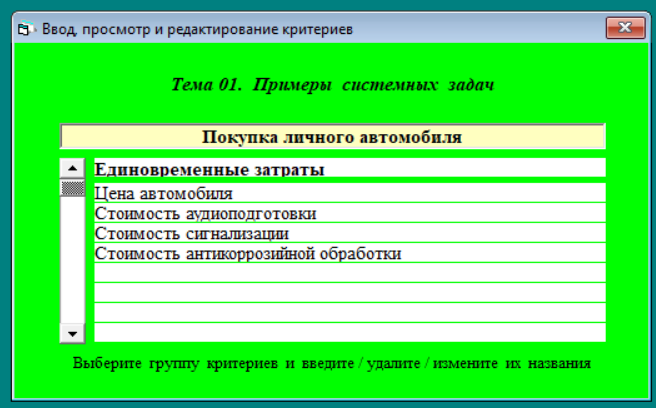


Рис. 13. Окно ввода критериев в программе Variant

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 14. Окно ввода вариантов в программе Variant

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 15. Сравнение групп, критериев и вариантов в программе Variant

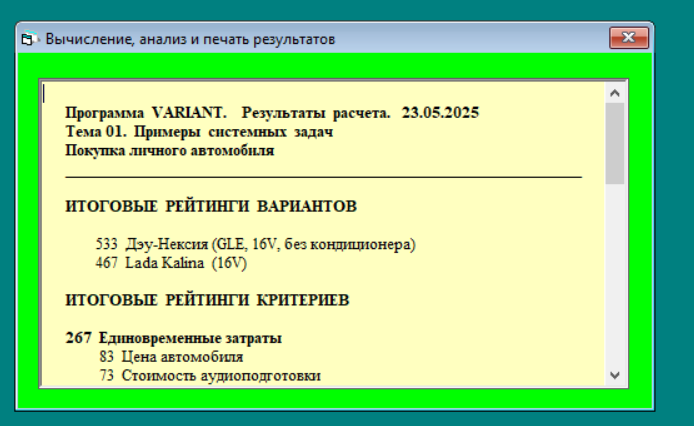


Рис. 16. Фрагмент итогового отчета в программе Variant

## Язык C++ и фреймворк Qt

В качестве языка программирования для программы было принято решение использовать язык C++.

C++ — компилируемый, статически типизированный язык программирования общего назначения.

Выбор обоснован высокой производительностью языка, что необходимо при работе с матрицами и позволяет разработать решение, доступное на разных платформах.

Так же стоит отметить, что в языке присутствует возможность создания классов, что значительно упрощает создание независимых модулей программы. Как раз метод анализа иерархий реализуем посредством класса, что будет показано далее.

Помимо языка разработки необходимо

В качестве основной программной платформы было принято решение использовать Qt.

Qt — это кросс-платформенный фреймворк () для разработки программного обеспечения. Он предоставляет инструменты и библиотеки для создания графических пользовательских интерфейсов (GUI), а также для разработки приложений сетевого и консольного взаимодействия, работы с базами данных, многопоточности и других задач

Фреймворк обладает следующими особенностями:

* **Кросс-платформенность**. Фреймворк существует для всех популярных операционных систем: Windows, Linux, MAC.
* **Поддержка различных языков программирования**. Разработчики могут выбирать наиболее удобный язык для реализации своих идей, а фреймворк обеспечит универсальное взаимодействие между компонентами и модулями приложения.
* **Модульность и масштабируемость**. Разработчики могут использовать только необходимые компоненты и модули фреймворка, что позволяет создавать лёгкие и оптимизированные приложения.
* **Работа с сетью, мультимедиа, базами данных**. Фреймворк обеспечивает широкие возможности для работы с этими аспектами, а также поддерживает интеграцию с внешними библиотеками и API

Qt выбран так как является открытым и позволяет создавать модульное, легко масштабируемое приложение, способное запускаться на разных операционных системах

Для фреймворка существует отдельная среда разработки под названием Qt Creator.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 17. Среда разработки Qt Creator

# Часть 2. Программная реализация

## Архитектура программы

Программа состоит из следующих модулей:

1. Main(main.cpp) – происходит инициализация приложение и вызов окна приложения.
2. Mainwindow – состоит из нескольких файлов
   1. mainwindow.ui – описывает весь интерфейс программы
   2. mainwindow.h и mainwindow.cpp – описывают всю логику взаимодействия с интерфейсом.
3. AppLog (appLog.h, appLog.cpp) – описывает дополнительные функции, что необходимы для работы программы но не взаимодействуют с интерфесом напрямую.
4. MAI (MAI.h, MAI.cpp) – в данном модуле описан класс, отвечающий за реализации метода анализа иерархий и возможность.

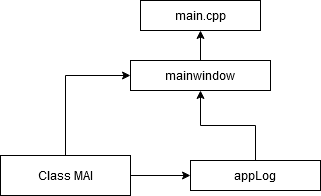


Рис. 18. Краткая архитектура приложения

Основным классом в приложении является MAI. Он отвечает за хранение списков критериев и вариантов, расчет, весов и итоговых вариантов, отправку и выгрузку данных из JSON файлов.

Класс заключает следующие поля:

1. QString taskName – имя задачи
2. QVector<QString> m\_criteria – хранит список критериев по которым производится срквнение вариантов
3. QVector<QString> m\_alternatives – массив со списком вариантов, которые необходимо сравнить
4. QVector<QVector<double>> m\_criteriaMatrix – матрица попарного сравнения критериев
5. QVector<QVector<QVector<double>>> m\_alternativeMatrices – вектор, хранящий матрицы сравненийц вариантов по критериям
6. QVector<double> m\_criteriaWeights – матрица весов критериев
7. QVector<QVector<double>> m\_alternativeWeights – вектор, хранимый массивы весов сравнения вариантов по критериям
8. QVector<double> m\_finalScores – вектор итоговых рейтингов вариантов

Все поля имеют модификатор доступа private, что необходимо для возможности доступа к данным только через методы классы, что обеспечивает инкапсуляцию данных.

Классы включает в следующие методы

Общедоступные методы (модификатор доступа public):

1. void setCriteria(const QVector<QString>& criteria) – принимает в качестве аргумента вектор имен критериев и заполняет список критериев класса
2. void setAlternatives(const QVector<QString>& alternatives) – принимает вектор имен критериев и заполняет соответствующий список критериев внутри класса
3. void setCriteriaPairwiseComparison(int row, int col, double value) – принимает в качестве параметров номера сравниваемых критериев (row и col) и значение сравнения критериев. Помимо этого, метод заполняет противоположный элемент списка обратным значением

Листинг 1. Метод setCriteriaPairwiseComparison

|  |
| --- |
| void MAI::setCriteriaPairwiseComparison(int row, int col, double value)  {  if (row >= 0 && row < m\_criteriaMatrix.size() && col >= 0 && col <m\_criteriaMatrix.size())  {  m\_criteriaMatrix[row][col] = value;  m\_criteriaMatrix[col][row] = 1.0 / value;  }  } |

1. void setAlternativePairwiseComparison(int criterionIndex, int row, int col, double value) – как и метод setCriteriaPairwiseComparison заполняет матрицу попарных сравнений, но для вариантов с заданием номера критерия по которому происходит сравнение. В качестве аргументов принимает номер критерия, по которому происходит сравнение, номера сравниваемых вариантов и значение сравнения
2. bool calculateWeights() – производит расчет весов критериев и альтернатив по критериям. В случае успеха выводит true, иначе false
3. QVector<double> getCriteriaWeights() – возвращает матрицу весов критериев
4. QVector<QVector<double>> getAlternativeWeights() – возвращает набор массивов с матрицами весов альтернатив по критериям
5. QVector<double> getFinalScores() – возвращает вектор с итоговым рейтингом вариантов
6. QString getWinAlter() – возвращает вариант с наибольшим рейтингом
7. QVector<QString> GetRateVars() – Возвращает рейтинг вариантов отсортированный по рейтингу
8. bool saveToJson(const QString& filename) – создает файл JSON и записывает в него информацию о задаче. В случае ошибки возвращает false.
9. bool loadFromJson(const QString& filename) – производит загрузку задачи из существующего файла JSON, если файл не существует или возникает ишибка при чтении возвращает false.

Помимо описанных методов существует еще два метода с модификатором досмтупа private. Они необходимо для внутренних расчетов класса.

1. void calculateFinalScores() – подсчитывает рейтинг альтернатив
2. bool calculateMatrixWeights(const QVector<QVector<double>>& matrix, QVector<double>& weights) – подсчитывает веса критериев и альтернатив. Выполняет предворительные расчеты перед вызовом метода calculateFinalScores

## Организация хранения данных

Для хранения данных было принято решение использовать формат JSON.

JSON (JavaScript Object Notation) — стандартный текстовый формат для хранения и передачи структурированных данных. Он основан на синтаксисе объекта в JavaScript, но не привязан к нему.

Формат выбран по причине того, что формат достаточно популярен, имеет удобную библиотеку для ввода/вывода и удобен для чтения пользователем, что позволяет анализировать данные без запуска самой программы .

JSON файл имеет следующую структуру:

1. Поле “name” – хранит имя задачи
2. Массив “alternatives” – хранит набор строк с вариантами выбора
3. Массив “criteria” – содержит набор строк с критериями оценивания вариантов
4. Массив “criteria\_matrix” – матрица соотношения критериев между собой
5. Массив “alternative\_matrices” – хранит набор матриц сравнения альтернатив по каждому критерию

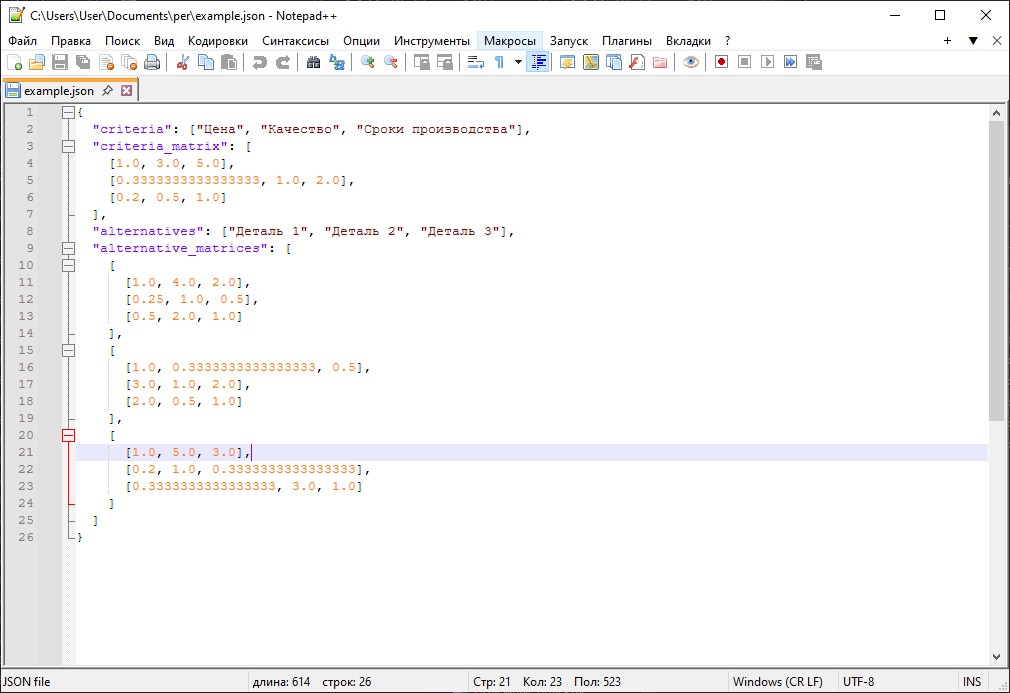


Рис. 19. Представление информации в Notepad++

## Интерфейс программы

Интерфейс программы разделен на несколько вкладок и общую часть.

В верхней части (общей части) программы представлены 3 кнопки:

1. Кнопка «Открыть файл»  – открывает существующий JSON файл с задачей, используя диалоговое окно «Открыть файл» (рис 20)

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 20. Диалоговое окно «Открыть файл»

1. Кнопка «Сохранить как»  – сохраняет данные в новый JSON файл с использованием диалогового окна «Сохранить как» (рис. 21)

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 21. Диалоговое окно «Сохранить как»

1. Кнопка «Сохранить»  – сохраняет данные в существующий JSON файл. Если файл новый запускается функционал кнопки «Сохранить как». Если в процессе работы с существующим файлом пользователь меняет имя задачи(файла) программа предлагает переименовать существующий файл. Если пользователь отказывается, то программа создает файл с новым именем

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 22. Вкладка вода вариантов

В разделе «Варианты» (рис 22) пользователь вводит имя задачи. Так же имя задачи является именем файла JSON в котором хранится существующая информация о задаче (варианты, критерии, весовые матрицы)

Изображение выглядит как текст, снимок экрана, дисплей, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 23. Вкладка ввода критериев

В разделе «Критерии» (рис 23) вводятся критерии, по которым производится сравнение вариантов и заполняется матрица сравнения критериев. Критерии сравниваются в диапазоне от 9 до (0.11). 9 – намного лучше. 1 – одинаковы. – намного хуже. При вводе значений в ячейки в противоположное положение автоматически вводится обратное значение. При вводе некорректного значения в противоположную ячейку вводится значение   
-1, что говорит об ошибке.

При нажатии на кнопку  выводится справка о сравнении критериев.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 24. Вкладка сравнения вариантов по критериям

На вкладке «Сравнения» (рис 24) производится сравнения вариантов по критериям. Сравнение вариантов между собой производится аналогично со сравнением критериев между собой. Главным отличием является то, что сравнение производится по каждому критерию. Критерий необходимый для сравнения выбирается в выпадающем списке «Критерии». В процессе сравнения необходимо так же, как и при сравнении критериев заполнить матрицу попарных сравнений. Диапазон значений – от 9 до . И аналогично со сравнением критериев при вводе некорректного значения в текущую ячейку и противоположную вводится значение -1, что говорит об ошибке ввода.

При нажатии на кнопку  выводится справка о сравнении вариантов по критериям.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 25. Вкладка выводов результатов сравнения

В разделе «Результат» (рис 25) по нажатии кнопки «Произвести сравнение» производит расчет результатов и выводит итоговый результат с рейтингов альтернатив. Рейтинг выводится с целью определения на сколько оптимальный вариант лучше остальных.

Каждая вкладка приложения является отдельным этапом. Все этапы необходимо проходить последовательно в следующем порядке: Варианты 🡪 Критерии 🡪 Сравнения 🡪 Результат.

К следующему этапу необходимо подходить только по завершению предыдущего этапа.

# Часть 3. Пример использования

## Постановка задачи

## Пример решения

## Сравнение с другими способами решения задачи

## Возможные улучшения

# Приложение A Файл MAI.h Заголовочный файл метода анализа иерархий

#ifndef MAI\_H

#define MAI\_H

#include <QObject>

#include <QVector>

#include <QString>

#include <QJsonObject>

class MAI: public QObject

{

Q\_OBJECT

public:

explicit MAI(QObject \*parent = nullptr);

// Установка критериев и альтернатив

void setCriteria(const QVector<QString>& criteria);

void setAlternatives(const QVector<QString>& alternatives);

// Установка парных сравнений

void setCriteriaPairwiseComparison(int row, int col, double value);

void setAlternativePairwiseComparison(int criterionIndex, int row, int col, double value);

// Расчет весов

bool calculateWeights();

// Получение результатов

QVector<double> getCriteriaWeights() const;

QVector<QVector<double>> getAlternativeWeights() const;

QVector<double> getFinalScores() const;

// Работа с JSON

bool saveToJson(const QString& filename);

bool loadFromJson(const QString& filename);

private:

// Вспомогательные методы

bool calculateMatrixWeights(const QVector<QVector<double>>& matrix, QVector<double>& weights);

void calculateFinalScores();

// Данные

QVector<QString> m\_criteria;

QVector<QString> m\_alternatives ;

QVector<QVector<double>> m\_criteriaMatrix;

QVector<QVector<QVector<double>>> m\_alternativeMatrices;

// Результаты

QVector<double> m\_criteriaWeights;

QVector<QVector<double>> m\_alternativeWeights;

QVector<double> m\_finalScores;

};

#endif // MAI\_H

# Приложение Б Файл MAI.cpp Исполняемый файл метода анализа иерархий

#include "MAI.h"

#include <QJsonArray>

#include <QJsonDocument>

#include <QFile>

#include <cmath>

MAI::MAI(QObject \*parent) : QObject(parent) {}

void MAI::setCriteria(const QVector<QString>& criteria) {

m\_criteria = criteria;

m\_criteriaWeights.resize(criteria.size());

m\_criteriaMatrix.resize(criteria.size(), QVector<double>(criteria.size(), 1.0));

}

void MAI::setAlternatives(const QVector<QString>& alternatives) {

m\_alternatives = alternatives;

m\_alternativeMatrices.resize(m\_criteria.size());

for (auto& matrix : m\_alternativeMatrices) {

matrix.resize(alternatives.size(), QVector<double>(alternatives.size(), 1.0));

}

}

void MAI::setCriteriaPairwiseComparison(int row, int col, double value) {

if (row >= 0 && row < m\_criteriaMatrix.size() && col >= 0 && col < m\_criteriaMatrix.size()) {

m\_criteriaMatrix[row][col] = value;

m\_criteriaMatrix[col][row] = 1.0 / value;

}

}

void MAI::setAlternativePairwiseComparison(int criterionIndex, int row, int col, double value) {

if (criterionIndex >= 0 && criterionIndex < m\_alternativeMatrices.size() &&

row >= 0 && row < m\_alternativeMatrices[criterionIndex].size() &&

col >= 0 && col < m\_alternativeMatrices[criterionIndex].size()) {

m\_alternativeMatrices[criterionIndex][row][col] = value;

m\_alternativeMatrices[criterionIndex][col][row] = 1.0 / value;

}

}

bool MAI::calculateWeights() {

if (!calculateMatrixWeights(m\_criteriaMatrix, m\_criteriaWeights)) {

return false;

}

m\_alternativeWeights.resize(m\_alternativeMatrices.size());

for (int i = 0; i < m\_alternativeMatrices.size(); ++i) {

m\_alternativeWeights[i].resize(m\_alternatives .size());

if (!calculateMatrixWeights(m\_alternativeMatrices[i], m\_alternativeWeights[i])) {

return false;

}

}

calculateFinalScores();

return true;

}

QVector<double> MAI::getCriteriaWeights() const {

return m\_criteriaWeights;

}

QVector<QVector<double>> MAI::getAlternativeWeights() const {

return m\_alternativeWeights;

}

QVector<double> MAI::getFinalScores() const {

return m\_finalScores;

}

bool MAI::saveToJson(const QString& filename) {

QJsonObject root;

QJsonArray criteriaArray;

for (const auto& criterion : m\_criteria) {

criteriaArray.append(criterion);

}

root["criteria"] = criteriaArray;

QJsonArray criteriaMatrixArray;

for (const auto& row : m\_criteriaMatrix) {

QJsonArray rowArray;

for (double val : row) {

rowArray.append(val);

}

criteriaMatrixArray.append(rowArray);

}

root["criteria\_matrix"] = criteriaMatrixArray;

QJsonArray alternativesArray;

for (const auto& alternative : m\_alternatives ) {

alternativesArray.append(alternative);

}

root["alternatives"] = alternativesArray;

QJsonArray alternativeMatricesArray;

for (const auto& matrix : m\_alternativeMatrices) {

QJsonArray matrixArray;

for (const auto& row : matrix) {

QJsonArray rowArray;

for (double val : row) {

rowArray.append(val);

}

matrixArray.append(rowArray);

}

alternativeMatricesArray.append(matrixArray);

}

root["alternative\_matrices"] = alternativeMatricesArray;

QJsonDocument doc(root);

QFile file(filename);

if (!file.open(QIODevice::WriteOnly)) {

return false;

}

file.write(doc.toJson());

file.close();

return true;

}

bool MAI::loadFromJson(const QString& filename) {

QFile file(filename);

if (!file.open(QIODevice::ReadOnly)) {

return false;

}

QByteArray data = file.readAll();

file.close();

QJsonDocument doc = QJsonDocument::fromJson(data);

if (doc.isNull()) {

return false;

}

QJsonObject root = doc.object();

QJsonArray criteriaArray = root["criteria"].toArray();

m\_criteria.clear();

for (const auto& item : criteriaArray) {

m\_criteria.append(item.toString());

}

QJsonArray criteriaMatrixArray = root["criteria\_matrix"].toArray();

m\_criteriaMatrix.clear();

for (const auto& rowItem : criteriaMatrixArray) {

QJsonArray rowArray = rowItem.toArray();

QVector<double> row;

for (const auto& valItem : rowArray) {

row.append(valItem.toDouble());

}

m\_criteriaMatrix.append(row);

}

QJsonArray alternativesArray = root["alternatives"].toArray();

m\_alternatives .clear();

for (const auto& item : alternativesArray) {

m\_alternatives .append(item.toString());

}

QJsonArray alternativeMatricesArray = root["alternative\_matrices"].toArray();

m\_alternativeMatrices.clear();

for (const auto& matrixItem : alternativeMatricesArray) {

QJsonArray matrixArray = matrixItem.toArray();

QVector<QVector<double>> matrix;

for (const auto& rowItem : matrixArray) {

QJsonArray rowArray = rowItem.toArray();

QVector<double> row;

for (const auto& valItem : rowArray) {

row.append(valItem.toDouble());

}

matrix.append(row);

}

m\_alternativeMatrices.append(matrix);

}

return calculateWeights();

}

bool MAI::calculateMatrixWeights(const QVector<QVector<double>>& matrix, QVector<double>& weights) {

if (matrix.isEmpty()) return false;

int size = matrix.size();

weights.resize(size);

QVector<double> geometricMeans(size, 1.0);

for (int i = 0; i < size; ++i) {

for (int j = 0; j < size; ++j) {

geometricMeans[i] \*= matrix[i][j];

}

geometricMeans[i] = std::pow(geometricMeans[i], 1.0 / size);

}

double sum = 0.0;

for (double mean : geometricMeans) {

sum += mean;

}

if (sum == 0.0) return false;

for (int i = 0; i < size; ++i) {

weights[i] = geometricMeans[i] / sum;

}

return true;

}

void MAI::calculateFinalScores() {

m\_finalScores.resize(m\_alternatives .size(), 0.0);

for (int alt = 0; alt < m\_alternatives .size(); ++alt) {

for (int crit = 0; crit < m\_criteria.size(); ++crit) {

m\_finalScores[alt] += m\_criteriaWeights[crit] \* m\_alternativeWeights[crit][alt];

}

}

}

# Приложение В Пример оформления JSON файла

{

  "criteria": ["Цена", "Качество", "Сроки производства"],

  "criteria\_matrix": [

    [1.0, 3.0, 5.0],

    [0.3333333333333333, 1.0, 2.0],

    [0.2, 0.5, 1.0]

  ],

  "alternatives": ["Деталь 1", "Деталь 2", "Деталь 3"],

  "alternative\_matrices": [

    [

      [1.0, 4.0, 2.0],

      [0.25, 1.0, 0.5],

      [0.5, 2.0, 1.0]

    ],

    [

      [1.0, 0.3333333333333333, 0.5],

      [3.0, 1.0, 2.0],

      [2.0, 0.5, 1.0]

    ],

    [

      [1.0, 5.0, 3.0],

      [0.2, 1.0, 0.3333333333333333],

      [0.3333333333333333, 3.0, 1.0]

    ]

  ]

}