

Оглавление

[**Введение** 3](#_Toc127422559)

[**Глава 1. ОБЩАЯ ХАРАКТЕРИСТИКА И ЦЕЛИ АНАЛИЗА ВЕБ-РЕСУРСОВ** 5](#_Toc127422560)

[1.1. Общие понятия и сведения 5](#_Toc127422561)

[1.2. Архитектура веб-ресурсов 8](#_Toc127422562)

[1.3. Поисковая оптимизация 10](#_Toc127422563)

[1.4. Виды оптимизации: 10](#_Toc127422564)

[1.5. Статистика использования сети Интернет 14](#_Toc127422565)

[**Глава 2. ПОСТАНОВКА ЗАДАЧИ И ОПИСАНИЕ МАТЕМАТИЧЕСКИХ МЕТОДОВ И МОДЕЛЕЙ ОЦЕНКИ ЭФФИКТИВНОСТИ ВЕБ-РЕСУРСОВ** 16](#_Toc127422566)

[2.1 Математическая модель 16](#_Toc127422567)

[2.2 Расстояние Левенштейна 17](#_Toc127422568)

[2.3 Метод анализа иерархий 20](#_Toc127422569)

[2.4 Расстояние Хэмминга 25](#_Toc127422570)

[**Глава 3. РАЗРАБОТКА АЛГОРИТМА ДЛЯ ОЦЕНКИ ЭФФИКТИВНОСТИ ВЕБ-РЕСУРСОВ** 27](#_Toc127422571)

[3.1 Выбор инструментов 27](#_Toc127422572)

[3.2 Получение входных данных 28](#_Toc127422573)

[3.3 Выбор веб-ресурсов для анализа 29](#_Toc127422574)

[3.4 Сбор критериев 30](#_Toc127422575)

[3.5 Обработка данных для анализа 31](#_Toc127422576)

[3.6 Разработка методов для анализа полученных данных 32](#_Toc127422577)

[3.7 Анализ данных 33](#_Toc127422578)

[3.7.1 Расстояние Левенштейна 33](#_Toc127422579)

[3.7.2 Метод анализа иерархий 35](#_Toc127422580)

[**Заключение** 41](#_Toc127422581)

[**Список использованных источников** 42](#_Toc127422582)

[**Приложение А** 44](#_Toc127422583)

# **Введение**

В настоящее время каждой организации требуется собственный веб-ресурс для привлечения новых клиентов, продвижения собственного товара или предоставления услуг. Несмотря на это перед всеми стоит задача продвижения сайта его в списке выдачи поисковых систем (Google, Yandex и пр.) для привлечения большего количества пользователей, что подтверждает актуальность предмета анализа.

Продвижение веб-ресурса в списке выдачи поисковых систем является трудоемкой задачей и затратной, с точки зрения времени. Чтобы достигнуть этой цели необходимо провести анализ, определить вектор движения на основе полученных данных, а затем постепенно вносить корректировки и следить за их результатом.

Данная работа рассматривает внутреннюю и внешнюю SEO-оптимизацию конкретных страниц при помощи математических методов и моделей для оценки по критериям, которые оценивают роботы поисковых систем с реализацией этих методов на языке программирования Python.

**Объектом анализа данной работы** являются показатели релевантности веб-ресурсов, учитывающиеся роботами поисковых систем для продвижения в списке выдачи.

**Предметом анализа –** математические методы и модели, определяющие значения показателей релевантности.

**Основная цель анализа** – расчет показателей релевантности веб-страниц для выявления некорректных технических решений с точки зрения продвижения в списке выдачи поисковых систем.

**Для достижения цели в работе предоставлены и решены следующие задачи:**

1. Сбор критериев оценки веб-ресурсов;
2. Построение математических моделей;
3. Написание программы на языке программирование Python для сбора и анализа данных.

**Используемое программное обеспечение:** Python, Sublime Text 3.

**Выпускная работа содержит следующую структуру:** введение, три главы, заключение, список использованных источников и приложения.

**В первой главе** рассматриваются веб-ресурсы, их архитектура, тенденции развития, а также способы оптимизации.

**Во второй главе** описываются математические методы и модели, оценивающие релевантность веб-страниц.

**Третья глава** описывает алгоритм оценки веб-ресурсов.

# **Глава 1. ОБЩАЯ ХАРАКТЕРИСТИКА И ЦЕЛИ АНАЛИЗА ВЕБ-РЕСУРСОВ**

## 1.1. Общие понятия и сведения

На сегодняшний день веб-ресурсы стали играть важную роль источника информации о предоставляемых организацией товарах или услугах. Грамотная реализация и структура позволяет не только привлечь новых клиентов, но и повысить собственные доходы за их счет.

Так как Интернет является одним из самых эффективных средств реализации взаимодействия с клиентами, быстрым и экономичным способом распространения информации о деятельности организации, внедрение веб-ресурса стало одним из приоритетных задач продвижения.

Перед каждой организацией, продвигающей или рекламирующей свой товар с помощью веб-ресурсов, становится задача привлечения новых клиентов. Для выполнения данной цели необходимо, чтобы большее количество пользователей посещало веб-ресурс, следовательно, необходимо чтобы данный веб-ресурс был как можно выше в списке выдачи поисковыми системами.

Согласно статистике Google [28]:

* 0.44% пользователей переходит на вторую страницу выдачи поисковой системы;
* 9% пользователей опускается до конца первый страницы выдачи;
* 59% пользователей посещает только одну страницу из поиска. 6% -  четыре и более страниц.

Приведенные выше статистические данные показывают важность продвижения веб-ресурса в лидирующие позиции поисковой выдачи. Исходя из вышесказанного, можно сделать вывод, что позиция веб-ресурса сильно влияет на количество посетителей (клиентов).

**Веб-ресурс** – это любой идентифицируемый ресурс, присутствующий в сети Интернет или подключенный к ней. Ресурсы идентифицируются с помощью единственных идентификаторов ресурсов (URI). В Семантической паутине веб-ресурсы и их семантические свойства описываются с помощью структуры описания ресурсов (RDF) [2].

**URI** – это последовательность символов, которая идентифицирует веб-ресурс, а также позволяет интернет-протоколам облегчить взаимодействие между этими ресурсами [3]. Такая последовательность представляет собой простой, расширяемый способ идентификации интернет-ресурсов. Благодаря единообразию, которое обеспечивают URI, различные типы идентификаторов ресурсов могут использоваться в одном и том же контексте, независимо от механизмов, используемых для доступа к этим ресурсам.

Идентификаторы ресурсов также могут быть повторно использованы в различных контекстах.

URI могут идентифицировать различные типы ресурсов, включая:

1. электронные документы;
2. веб-страницы;
3. изображения;
4. источники информации с единой целью.

**Семантическая паутина** [4] является расширением Всемирной паутины с помощью стандартов, установленных Консорциумом Всемирной паутины (W3C). Цель Семантической паутины – сделать данные Интернета машиночитаемыми.

Чтобы обеспечить возможность кодирования семантики данных, используются такие технологии, как Resource Description Framework (RDF) [21] и Web Ontology Language (OWL) [22]. Эти технологии используются для формального представления метаданных. Например, онтология может описывать понятия, отношения между сущностями и категории вещей. Эта встроенная семантика дает значительные преимущества, такие как рассуждения над данными и работа с разнородными источниками данных.

Эти стандарты способствуют развитию общих форматов данных и протоколов обмена в сети Интернет, в частности, RDF. Согласно W3C, «Семантическая паутина обеспечивает общую структуру, которая позволяет обмениваться данными и повторно использовать их через границы приложений, предприятий и сообществ», поэтому он рассматривается как интегратор различных информационных приложений и систем [1].

**Семантические свойства** (или свойства значения) [23] – это те аспекты языковой единицы, такой как морфема, слово или предложение, которые вносят вклад в значение этой единицы. К основным семантическим свойствам относятся значимость (или бессмысленность) Помимо самого выражения, существуют семантические отношения более высокого уровня, которые описывают связь между единицами: к ним относятся синонимия, антонимия и гипонимия.

Помимо основных свойств семантики, семантическое свойство также иногда используется для описания семантических компонентов слова. В этом смысле семантические свойства используются для определения семантического поля слова или набора слов.

**RDF** – это стандартная модель для обмена данными в сети Интернет. Она обладает свойствами, которые облегчают объединение данных, даже если схемы, лежащие в их основе, отличаются, и специально поддерживает эволюцию схем с течением времени, не требуя изменения всех потребителей данных.

RDF расширяет структуру связей в Семантической паутине, используя URI для обозначения отношений между объектами, а также двух концов связи. Использование этой модели позволяет смешивать, раскрывать и совместно использовать структурированные и полуструктурированные данные.

Также эта структура связей образует направленный, маркированный граф, где ребра представляют именованную связь между двумя ресурсами, представленными узлами графа. Такое представление графа является самой простой мысленной моделью для RDF и часто используется в простых для понимания визуальных объяснениях.

## 1.2. Архитектура веб-ресурсов

Архитектура веб-ресурса – это иерархическая структура веб-страниц. Такая инфраструктура отражается через внутреннюю перелинковку [24]. Структура веб-ресурса должна помогать пользователям находить информацию, а также указывать поисковым системам на взаимосвязь между различными страницами.

Типичная структура веб-ресурса выглядит как древовидный граф с корнями, в котором главная страница – корень. Веб-страницы, на которые ведут ссылки с главной страницы, являются ветвями, и от каждой страницы отходят дополнительные ветви.

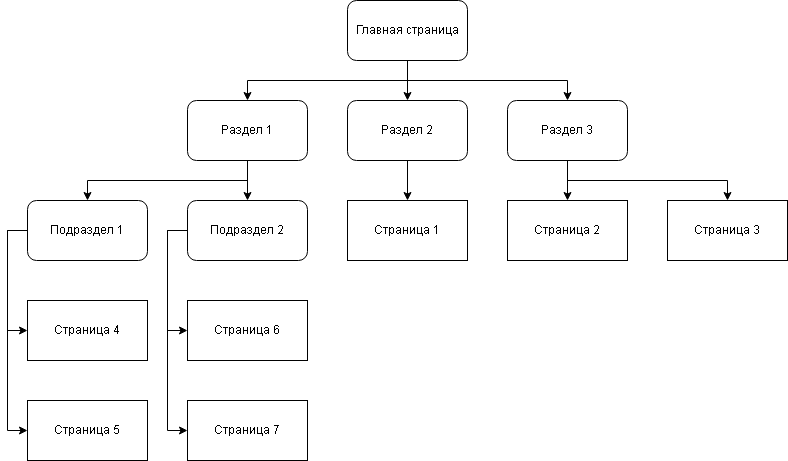


Рис. 1.1. Представление древовидной структуры веб-ресурса.

Продуманная архитектура веб-ресурса повышает удобство пользования сайтом. При выстраивании структуры сайта интуитивно понятным образом, пользователи могут легко найти нужную им информацию.

Кроме того, когда пользовательский опыт высок, повышается и рейтинг в поисковых системах. Пользователи будут проводить больше времени на вашем сайте и ссылаться на ваши веб-страницы, что является серьезным показателем того, что ваш продукт обретает целевую аудиторию, покупателей, читателей и т.д.

Также надежная архитектура сайта:

1. Помогает поисковым системам эффективно просматривать   
   веб-ресурс;
2. Поощряет глубокую навигацию по сайту, предоставляя пользователям больше страниц для посещения;
3. Распределяет «авторитет страницы» более справедливо, чтобы ни одна страница не осталась без внимания;
4. Укрепляет тематический авторитет благодаря сильной структуре внутренних ссылок, между связанными или похожими темами;
5. Повышает конверсию, облегчая пользователям поиск продуктов.

Выбор архитектуры конкретного веб-ресурса опирается на потребности пользователей или бизнес цели, например:

* «Сверху вниз» – где основное внимание уделяется общим категориям содержимого веб-страницы. Также такой подход помогает логически разделить контент, постепенно разбивая его на категории;
* «Снизу вверх» противоположен подходу «сверху вниз». При такой архитектуре основное внимание уделяется созданию структуры на основе контента, имеющегося на веб-ресурсе, путем группировки элементов в категории самого низкого уровня и, в свою очередь, группировки этих категорий в категории более высокого уровня.

## 1.3. Поисковая оптимизация

Поисковая оптимизация (SEO) – это процесс улучшения качества и количества трафика на веб-ресурс или веб-страницу из поисковых систем [1]. SEO нацелена на неоплаченный трафик (известный как «естественный» или «органический»), а не на прямой или платный. Неоплаченный трафик может поступать из различных видов поиска, включая поиск изображений, поиск видео, поиск новостей и отраслевые вертикальные поисковые системы.

Как стратегия интернет-маркетинга, SEO учитывает принцип работы поисковых систем, которые диктуют поведение поисковых систем, то, что ищут пользователи, фактические поисковые термины или ключевые слова, вводимые в поисковые системы, и то, какие поисковые системы предпочитает целевая аудитория. Поисковая оптимизация выполняется для привлечения новых посетителей из поисковых систем в связи с более высокой позицией на страницах результатов поисковых систем (SERP). Эти посетители потенциально могут быть преобразованы в клиентов.

SERP (страница результатов поисковой системы) – эта веб-страница создается поисковой системой в ответ на определяемый пользователем поисковый запрос. Интерфейс и функциональные компоненты являются стабильными характеристиками для конкретной поисковой системы и остаются неизменными во времени. Однако внешний вид выдачи Google меняется в каждом конкретном случае, так как большая часть органического вывода и динамических рекламных блоков контекстной рекламы – это платные ссылки, компактные текстовые фрагменты, которые размещаются в результатах поиска на платной основе. Оставшаяся область зарезервирована для необязательных элементов, которые различаются в зависимости от введенного запроса:

## 1.4. Виды оптимизации:

**Внутренняя оптимизация** веб-страниц – это комплекс мер, направленных на улучшение восприятия ресурса поисковыми системами и пользователями [25]. Она включает оптимизацию кода и структуры веб-ресурса, работу с контентом и коммерческими факторами, анализ и улучшение юзабилити. Включает в себя:

- белую (естественную) оптимизацию;

- серую оптимизацию;

- чёрную оптимизацию.

**Белая оптимизация** подразумевает анализ действий потребителей (клиентов веб-ресурса) на отдельных веб-страницах для достижения максимальной эффективности – возрастания числа посещений, популярности среди пользователей, а также рейтинга в поисковых системах. Такая оптимизация исключает все методы оптимизации, которые могут привести к блокировке веб-ресурса, в отличии от черной оптимизации. Внутренняя белая оптимизация включает в себя работу над:

1. Видимостью веб-ресурса для роботов поисковых систем;
2. Удобством для посетителей;
3. Качеством обработки заявок;
4. Совершенствованием контента для формирования семантического ядра;
5. Запросами, которые относятся к продвигаемому продукту;
6. Поиском партнерских программ.

Немаловажную роль играет подбор и размещение meta-тегов в коде страницы сайта (краткого описания веб-страницы). Этот текст будет виден пользователю в списке выдачи поисковых систем (Рис. 1.4.), а слова, или словосочетания, будут выделены жирным шрифтом (Рис. 1.5.).



Рис. 1.4. Meta-тег с описанием страницы.



Рис. 1.5. Список выдачи поисковой системы.

Немаловажно составить текст веб-страницы таким образом, чтобы ключевые слова встречались в нем равномерно по всему тексту. Однако стоит отметить, что переизбыток таких слов может навредить как пользователю, из-за плохой читаемости содержимого, так и самому веб-ресурсу, из-за расценивания поисковыми роботами как спам.

Внешняя белая оптимизация представляет собой добавление веб-ресурса в базы поисковых систем. Этот способ помогает поисковым роботам проверять данный веб-ресурс и добавлять его в результаты выдачи.

**Серая оптимизация** – это способ оптимизации, использующий большое количество ключевых слов в тексте веб-страницы. Такой подход может негативно сказаться на дальнейшем продвижении в связи с санкциями, которые накладывают поисковые системы, однако эффективен для поискового продвижения.

**Чёрная оптимизация** использует способы манипуляции результатами поиска для быстрого продвижения сайта. Основными методами служат:

1. Наполнение meta-тегов популярными, в поисковых запросах, ключевыми словами;
2. Переполнение текста веб-страницы ключевыми словами для искусственного увеличения веса ключевого слова;
3. Использование невидимого текста, но индексируемого поисковыми роботами;
4. Использование спам-ссылок, увеличивающих параметр «link popularity».
   1. Создание небольших сайтов с большим количеством тематических каталогов, ссылающихся на основной веб-ресурс;
   2. Участие в обмене ссылками;
   3. Приобретение ссылок на других веб-ресурсах;
   4. Ссылочный спам в социальных сетях, блогах и т.д.

Для противодействия такому подходу, поисковые машины имеют базу, куда заносятся сайта, с которых не учитывается ссылочное ранжирование.

К черной оптимизации также относятся:

Дорвеи – это разновидность поискового спама, в котором веб-страница, специально ориентированная под конкретные ключевые слова, или словосочетания, и дальнейшего перенаправления посетителей на другие страницы веб-ресурса;

Маскировка – предоставление поисковой машине одного содержимого, а пользователю – другого;

Рерайтинг – размещение текста из других источников.

В случае обнаружения поисковыми роботами вышеуказанных действий, веб-ресурс может быть исключен из результатов поиска.

## 1.5. Статистика использования сети Интернет

Основываясь на статистических данных веб-ресурса Datareportal.com [26], предоставленных за 2022 год:

1. Во всем мире насчитывается 5,07 миллиарда пользователей сети Интернет – это 63,50% всего населения мира. По сравнению с прошлым годом, количество пользователей возрос на 171 миллион человек, что составляет 3,5% роста, относительно 2021 года;
2. 96,10% пользователей в возрасте от 16 до 64 лет имеют мобильный телефон. Из них 95,80% имеют смартфоны;
3. 92,10% пользователей выходят в интернет через собственные мобильные телефоны. Даже с учетом того факта, что некоторые владельцы мобильных телефонов не используют их для выхода в Интернет;
4. 59,00% интернет-пользователей в возрасте от 16 до 64 используют ноутбук или настольный компьютер;
5. Среднее пребывание пользователя в сети Интернет составляет 6 часов 37 минут в день. Более трети этого времени уходит на социальные сети;
6. Число уникальных пользователей мобильных телефонов составляет 5,48 миллиарда человек, что составляет 68,60% населения мира.

Приведенные выше статистические показатели показывают важность сети Интернет в повседневной жизни каждого человека. Исходя из вышесказанного, можно сделать вывод, что развитие, анализ и продвижение веб-ресурсов очень важно для составления конкуренции между схожими организациями, предоставляющие аналогичные услуги или работы с помощью сети Интернет.

Максимальную активность в цифровой среде проявляют городские жители в возрасте 15–24 лет: 97,3% из них используют Интернет ежедневно. Среди молодежи, проживающей в сельской местности, эта доля чуть ниже – 92,9%.

# **Глава 2. ПОСТАНОВКА ЗАДАЧИ И ОПИСАНИЕ МАТЕМАТИЧЕСКИХ МЕТОДОВ И МОДЕЛЕЙ ОЦЕНКИ ЭФФИКТИВНОСТИ ВЕБ-РЕСУРСОВ**

2.1 Математическая модель – это математическое представление сценария реального мира для прогнозирования или получения информации [12].

Суть математической модели – это динамическое моделирование, а не фиксированный способ мышления с использованием математических символов, формул, языков, графиков и т.д. для абстрагирования, обобщения и описания сути проблемы, направленные на объяснение некоторых объективных явлений и законов развития в жизни. Математическое моделирование требует от человека гибкого использования соответствующих знаний математики, а также тщательного наблюдения и анализа реальных проблем в той или иной области, абстрагирования от проблем, а также извлечения математической модели, что и называется математическим моделированием.

Процесс математического моделирования – это представление способа мышления от абстракции к образности, которое реализуется в реальном процессе моделирования с помощью цифр, букв и соответствий. В процессе математического моделирования, и решения с его помощью практических задач, существует разделение уровней и иерархические требования, основная блок-схема которых представлена на Рис. 2.1.

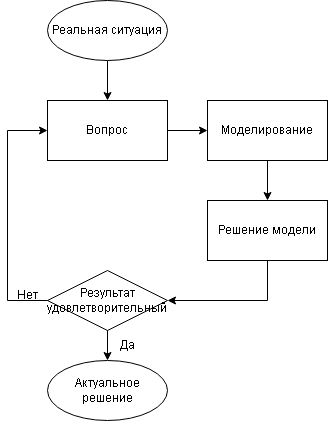


Рис. 2.1. Блок-схема иерархических требований.

## 2.2 Расстояние Левенштейна

В данной работе, для нахождения соотношений между блоками текста, использовался метод для измерения разницы между двумя последовательностями – **расстояние Левенштейна**. Метод получил название в честь советского математика Владимира Левенштейна, который рассмотрел это расстояние в 1965 году [5]. Эта метрика широко используются в теории информации, компьютерной лингвистике и информатике. Также можно использовать расстояние Левенштейна между двумя словами, чтобы получить информацию о минимальном количестве односимвольных изменения (замен, удалений, вставок) для изменения одного слова на другое.

Для определения Расстояния Левенштейна между двумя словами и задается функция :

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

– соответствует удалению;

– соответствует вставке;

– соответствует замене.

Рассмотрим пример анализа двух строк «Ерик» и «Крик».

Первым делом создается таблица, в которую вносятся исходные данные – символы строк, как показано в Таблице 2.1.

Таблица 2.1

Первый шаг определения расстояния Левенштейна

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **-** | **К** | **р** | **и** | **к** |
| **-** | **0** | **1** |  |  |  |
| **Е** | **1** |  |  |  |  |
| **р** |  |  |  |  |  |
| **и** |  |  |  |  |  |
| **к** |  |  |  |  |  |

Из примера выше видно, что для превращения символа «К» необходимо выполнить одно действие – удалить символ. Для символа «Е» проведем аналогичное действие.

Для символов «р» необходимо выполнить два действия удаления, следовательно, наша таблица примет вид (Таблица 2.2):

Таблица 2.2

Определение количества операций для полного удаления слов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | - | К | р | и | к |
| - | 0 | 1 | **2** | **3** | **4** |
| Е | 1 |  |  |  |  |
| р | **2** |  |  |  |  |
| и | **3** |  |  |  |  |
| к | **4** |  |  |  |  |

Рассчитаем значение для ячейки D(1, 1). Так как символы «К» и «Е» не равны друг другу, получим , тогда ячейка D(1, 1) является минимумом между значениями , и . Получаем функцию: .

Таблица 2.3

Определение количества шагов для замены символа «К» на «Е»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | - | К | р | и | к |
| - | 0 | 1 | 2 | 3 | 4 |
| Е | 1 | **1** |  |  |  |
| р | 2 |  |  |  |  |
| и | 3 |  |  |  |  |
| к | 4 |  |  |  |  |

Аналогичным способом заполним оставшиеся ячейки таблицы.

Таблица 2.4

Решенная таблица расстояния Левенштейна

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | - | К | р | и | к |
| - | 0 | 1 | 2 | 3 | 4 |
| Е | 1 | 1 | **2** | **3** | **4** |
| р | 2 | **2** | **1** | **2** | **3** |
| и | 3 | **3** | **2** | **1** | **2** |
| к | 4 | **4** | **3** | **2** | **1** |

Из Таблицы 4 видим, что для преобразования строки «Крик» в «Ерик» потребуется один шаг, который определили в ячейке D(1, 1).

## 2.3 Метод анализа иерархий

В теории принятия решений метод анализа иерархий (МАИ), представляет собой структурированную технику для организации и анализа сложных решений, основанную на математике и психологии [7]. Он был разработан Томасом Л. Саати в 1970-х годах; в 1983 году Саати совместно с Эрнестом Форманом разработал программное обеспечение Expert Choice, и с тех пор МАИ активно изучается и совершенствуется.

Этот метод представляет собой точный подход к количественной оценке весов критериев принятия решений. Опыт отдельных экспертов используется для оценки относительной значимости факторов путем попарных сравнений. Каждый из респондентов сравнивает относительную важность каждой пары элементов с помощью специально разработанного вопросника.

Метод анализа иерархий имеет особое применение в групповом принятии решений, и используется во всем мире в самых разных ситуациях принятия решений.

МАИ помогает аналитикам найти то решение, которое наилучшим образом соответствует их цели и их пониманию проблемы. Она обеспечивает всеобъемлющую и рациональную структуру для структурирования проблемы принятия решения, для представления и количественной оценки ее элементов, для соотнесения этих элементов с общими целями и для оценки альтернативных решений.

В основе данного метода лежит декомпозиция [27] проблемы на более простые блоки. Данный принцип предусматривает структурирование проблемы в виде иерархии, что является первым этапом применения МАИ. Иерархия считается полной, если каждый элемент заданного уровня связан со всеми элементами последующего уровня. Простейшая полная иерархия проблемы многокритериального выбора включает в себя три уровня (Рис. 2.2) [7].



Рис. 2.2. – Иерархия проблемы.

**Принцип сравнительных суждений.** Чтобы установить приоритеты критериев и получить оценки для альтернативных решений, в методе анализа иерархий используется метод парных сравнений – строятся матрицы парных сравнений , где , - это вес i-го элемента иерархии.

Следовательно, (главная диагональ матрицы равна 1), (матрица является обратносимметричной).

По каждой матрице определяется вектор локальных приоритетов и вычисляется индекс согласованности мнений аналитика.

**Принцип синтеза приоритетов.** Приоритеты синтезируются, начиная со второго уровня иерархии сверху вниз. Локальные приоритеты альтернатив умножаются по каждому элементу в соответствии с критериями. Таким образом, итоговой оценкой альтернативы в методе парных сравнений является вес альтернативы, вычисляемый как свертка весовых коэффициентов критериев (локальных критериев) всех уровней иерархии.

Алгоритм метода анализа иерархий делится на пять этапов:

1. Формирование иерархии целей;
2. Определение приоритетов;
3. Расчет локальных векторов приоритетов;
4. Проверка оценок аналитика на непротиворечивость;
5. Расчет приоритетов целей и мероприятий для иерархии в целом на основе синтеза локальных приоритетов.

**Фундаментальная шкала предпочтений** (Таблице 2.5) показывает вес важности критериев по отношению друг к другу.

Таблица 2.5

Фундаментальная шкала предпочтений

|  |  |
| --- | --- |
|  | **Пояснения** |
| **1** | Равная важность сравниваемых элементов иерархии. Оба сравниваемых элемента имеют одинаковую значимость для элемента более высокого уровня |
| **3** | Умеренное превосходство i-го элемента иерархии над j-ым. Говорит о немного большей значимости одного элемента по сравнению с другим |
| **5** | Существенное или сильное превосходство i-го элемента. Говорит о более высокой значимости одного элемента по сравнению с другим |
| **7** | Значительное превосходство i-го элемента над j-ым. Очень высокая значимость элемента |
| **9** | Очень значительное превосходство i-го элемента. Максимально возможное различие между двумя элементами |
| **2, 4, 6, 8** | Промежуточные степени превосходства |

Расчет локальных векторов приоритетов. Для каждой матрицы можно рассчитать локальные приоритеты сравниваемых элементов. Каждый элемент матрицы ставим в соответствии с геометрическим средним строки, на которой он находится.

На этом этапе можно увидеть, что наиболее значимые критерии будут иметь больший вес локального вектора приоритетов.

**Проверка органичности приоритетов.** Первым делом вычисляется индекс согласованности (ИС) суждений по каждой матрице.

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

n – размерность матрицы;

λmax можно рассчитать следующим образом:

* Суммируется каждый столбец матрицы парных сравнений;
* Сумма первого столбца умножается на первую компоненту локального вектора приоритетов, сумма второго столбца на вторую компоненту и т.д.;
* Полученные произведения суммируются.

Далее, для получения отношения согласованности, производится расчет по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

СС – случайная согласованность, которая получилась бы при случайной выборе суждений по фундаментальной шкале (1/9 … 9) для заданного значения (Таблица 2.6) [8].

Таблица 2.6

Случайная согласованность

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Размерность матрицы | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Случайная согласованность | 0 | 0 | 0,58 | 0,90 | 1,12 | 1,24 | 1,32 | 1,41 | 1,45 | 1,49 |

Если для конкретной матрицы окажется, что ОС > 0,1, то можно утверждать, что суждения аналитика, на основе которых заполнена исследуемая матрица, сильно рассогласованы.

Для **расчета приоритетов для всей иерархии** необходимо умножить локальные приоритеты альтернатив на соответствующие критерии уровня, а затем просуммировать по каждому элементу в соответствии с критериями. В результате определяются глобальные приоритеты альтернатив с учетом приоритетов критериев. Наиболее высокий рейтинг будет соответствовать альтернативе с наибольшим значением глобального приоритета.

Сравнивая полученные значения глобальных приоритетов, можно определить рейтинг всех альтернатив. Согласно оцениванию с помощью метода анализа иерархий предпочтение следует отдать именно той альтернативе, значение глобального приоритета которой будет выше.

## 2.4 Расстояние Хэмминга

Применяется в случае наличия множества слов, тогда задается правило для вычисления расстояния между словами. Данный метод можно сравнить с вычислением расстояния между точками в пространстве. Аналогичным образом можно реализовать сравнение между словами. Чем меньше расстояние, тем больше схожи выбранные слова и наоборот [11].

Рассмотрим пример сравнения слов, использую алфавит [0, 1] и длиной слова 3, тогда получим 8 вариантов слов: 000, 001, 010, 011, 100, 101, 110, 111.

Из данных слов можно построить трехмерный куб, на вершине которого будут расположены слова (Рис. 2.3):

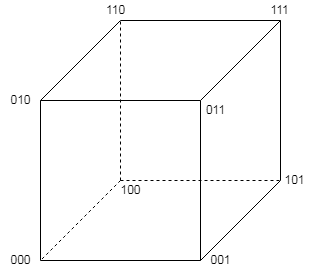


Рис. 2.3 – Визуальное представление расстояния Хэмминга.

Как видно из Рис. 2.3, максимальное расстояние между словами, например, 011 и 100 равно 3, для точек 010 и 001 расстояние будет равно 2, а 110 и 111 равно 1.

Так как выполняются все необходимые аксиомы (тождества, симметрии, неравенство треугольника), данная метрика является полноценным метрическим пространством.

Недостатком расстояния Хэмминга является:

* Обязательное наличие слов одинаковой длины.

Преимущества расстояния Хэмминга:

* Легко реализуемый алгоритм;
* Понятный способ вычислений;
* Точное измерение расстояния между строками.

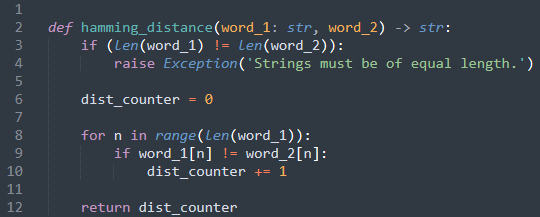


Рис. 2.4 – Пример реализации расстояния Хэмминга.

В связи с необходимость сравнения строк разных длин, в работе будет использоваться расстояние Левенштейна.

# **Глава 3. РАЗРАБОТКА АЛГОРИТМА ДЛЯ ОЦЕНКИ ЭФФИКТИВНОСТИ ВЕБ-РЕСУРСОВ**

Целью данной задачи является получение и анализ критериев, описывающих эффективность веб-станицы. Декомпозируем задачу для получения:

1. Выбор инструментов;
2. Получение входных данных;
3. Выбор веб-ресурсов для анализа;
4. Сбор критериев;
5. Обработка данных для анализа;
6. Разработка методов для анализа полученных данных;
7. Анализ данных.

## 3.1 Выбор инструментов

Для выполнения данной работы был выбран язык программирования Python, в связи с его удобным и понятным синтаксисом, простотой написания программного кода и его читаемостью.

Для получения html кода анализируемой страницы была выбрана библиотека requests [9]. Она позволяет отправлять на сервер запросы, а также получать ответ с них. В данной работе эта библиотека была применена для получения html кода анализируемой веб-страницы, а также отправки запросов к поисковой системе Google для получения результатов поиска по ключевым словам.

Получения html кода недостаточно, поэтому необходимо обработать его и получить необходимые для анализа данные. Этой задачей занимается библиотека bs4 [10], которая преобразовывает строку с html кодом в объект, с помощью которого можно получать теги, их атрибуты и хранимые в них данные с помощью поиска. Эта библиотека является одним из главных звеньев алгоритма т.к. информация, которая в дальнейшей будет анализироваться, предоставляет именно она.

## 3.2 Получение входных данных

Данные для анализа принимаются путем ввода пользователем необходимого запроса. Для удобного использования запросов разработаем класс, который будет хранить в себе введенные данные (Рис. 3.1).

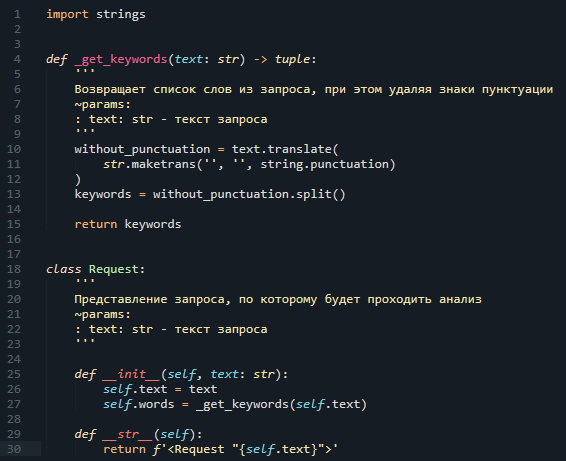


Рис. 3.1 – Код класса Request.

Приведенный выше класс Request принимает обязательный аргумент text, который будет храниться в нем. Также, в конструкторе данного класса, объявляется переменная *words*, которая хранит результат выполнения функции *\_get\_keywords*, возвращающая список слов запроса, удаляя все введенные знаки пунктуации, чтобы исключить возможность неправильного анализа.

## 3.3 Выбор веб-ресурсов для анализа

Для выбора веб-ресурсов реализуем функцию, которая будет отправлять запрос, введенный пользователем в класс *Request*, а затем возвращать 5 первых страниц из поисковой выдачи Google (Рис. 3.2).

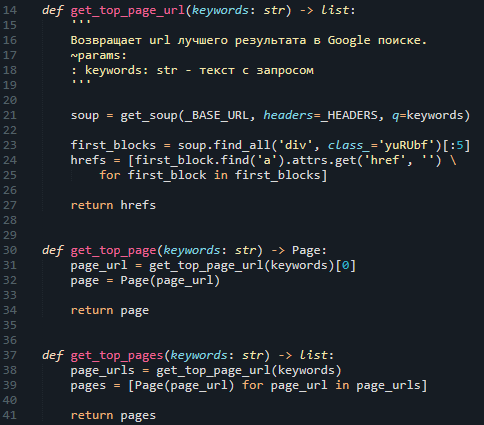


Рис. 3.2 – Реализация функции поиска первых страниц   
поисковой выдачи Google.

Для получения страниц была создана функция *get\_top\_pages*, которая, на основе полученных функцией *get\_top\_page\_url*, создает экземпляры класса Page.

Исходя из приведенного выше кода, проведем параллельный анализ эффективности веб-ресурсов для запроса «Изобретаем JPEG» (Рис. 3.3).

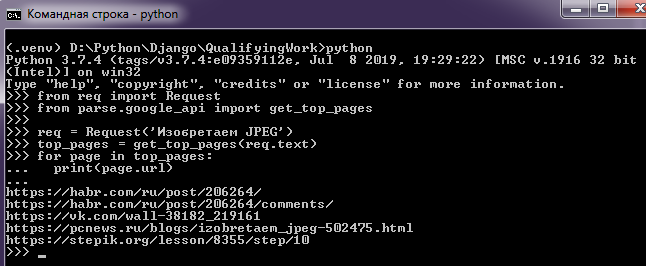


Рис. 3.3 – Полученные веб-страницы по запросу.

## 3.4 Сбор критериев

Теперь, имея url-адреса веб-страниц, можно приступать к сбору критериев. Создадим класс Page, который будет хранить все полученные результаты о данных страницы (Рис. 3.4).



Рис. 3.4 – Реализация класса Page.

Приведенный выше пример показывает, каким образом информация, полученная из html кода, находится и возвращается в виде данных.

## 3.5 Обработка данных для анализа

Обработка более труднодоступных данных была разделена на логические блоки, отвечающие за обработку текста (Рис. 3.5) и обработку тегов (Рис. 3.6):



Рис. 3.5 – Функции обработки текста.

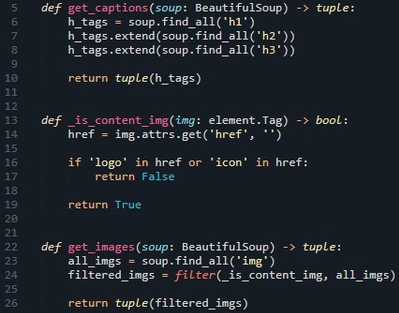


Рис. 3.6 – Функции обработки тегов.

Приведенные выше функции принимают обязательный аргумент – экземпляр класса *BeautifulSoup* (из библиотеки *bs4*), который предоставляет функции поиска html тегов на основе различных атрибутов.

## 3.6 Разработка методов для анализа полученных данных

Первым делом определим класс, который будет хранить значения следующих критериев (Рис. 3.7):

1. Соотношение текста из тега TITLE с первым текстом из тега H1;
2. Соотношение текста из тега TITLE с вводимым пользователем запросом;
3. Процент вхождения ключевых слов, из вводимого пользователем запроса, по отношению к основному тексту страницы;
4. Количество ключевых слов в тексте страницы;
5. Наличие вхождения запроса в первые 20% текста страницы;
6. Наличие ключевых слов, вводимые пользователем в запросе, в META-теге описания страницы (с атрибутом *name* равным *description*);
7. Наличие на странице хотя бы одной таблицы или перечисления (table, ol или ul);
8. Вхождение ключевых слов в тегах H1, H2, H3;
9. Наличие изображений на каждые 1 500 символов сплошного текста.

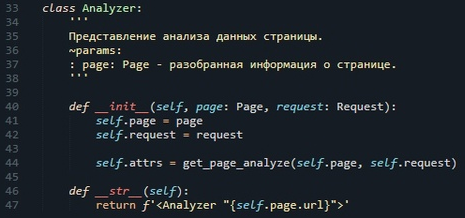


Рис. 3.7 – Реализация класса Analyzer.

## 3.7 Анализ данных

Так как большинство критериев опираются на соотношение строк в html тегах, был написан метод, определяющий эти показатели – **расстояние Левенштейна** (Рис. 3.8).

## 3.7.1 Расстояние Левенштейна

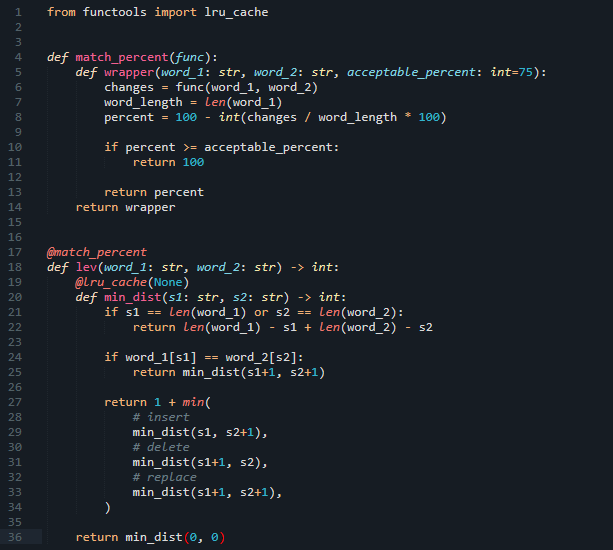


Рис. 3.8. Реализация расстояния Левенштейна.

Описанный выше метод определяет расстояния между двумя последовательностями и принимает на вход два аргумента – последовательности для сравнения. Внутри функции реализована вложенная рекурсивная функция , отвечающая за расчет совпадений. Входными аргументами функции являются два числа (индекса), указывающие на анализируемую ячейку таблицы, описанную в главе 2 данной работы. Изначально выполняется проверка на соответствие длин каждого слова с полученным индексом соответствующей последовательности. В случае, если проверка прошла, функция возвращает крайнее, на данный момент, количество изменений для соответствия текста друг другу, а в противном случае выполняется проверка на совпадение символов. Если они одинаковые, необходимо пройти дальше по главной диагонали, так как изменений в количестве шагов для преобразования нет. В случае, когда проверки, описанные выше, не прошли, находится минимум из рядом стоящих ячеек (как описано из формулы 1).

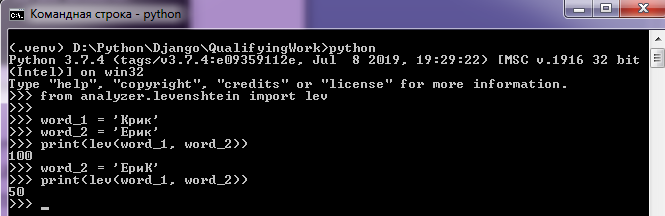


Рис. 3.9 – Пример использования расстояния Левенштейна.

Как видно из Рис. 3.8, функция *lev* обернута в декоратор *match\_percent*, который проверяет полученный результат и, в случае разницы между блоками текста больше 75%, округляет значение до 100.

С помощью этого метода были получены следующие критерии:

1. Соотношение текста из тега TITLE с первым текстом из тега H1;
2. Соотношение текста из тега TITLE с вводимым пользователем запросом;
3. Процент вхождения ключевых слов, из вводимого пользователем запроса, по отношению к основному тексту страницы;
4. Наличие вхождения запроса в первые 20% текста страницы;
5. Наличие ключевых слов, вводимые пользователем в запросе, в META-теге описания страницы (с атрибутом *name* равным *description*);
6. Вхождение ключевых слов в тегах H1, H2, H3.

## 3.7.2 Метод анализа иерархий

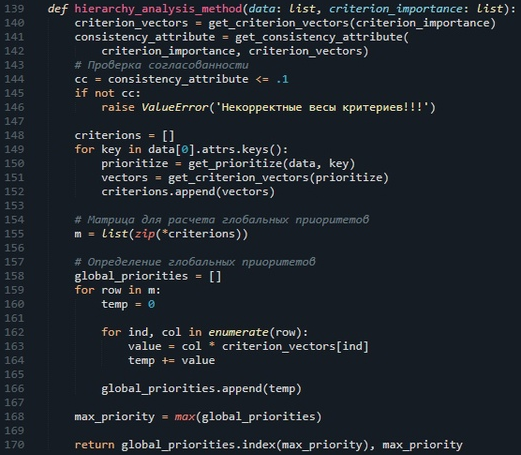


Рис. 3.10. Реализация метода анализа иерархий на языке   
программирования Python.

Описанная выше функция вычисляет наиболее релевантный вариант для выбора веб-ресурса, опираясь на критерии весов, которые определены с учетом важности параметров SEO. Весы релевантности записываются в квадратную матрицу, где производится оценка критериев друг к другу.

В качестве обязательного аргумента принимается матрица оценки важности критериев (Таблица 3.1).

В первую очередь определяются локальные векторы приоритетов, затем находим отношение согласованности, на основе чего определяем правильность заполнения входных критериев (ОС < 0.1). Если проверка прошла успешно, продолжаем работу функции и определяем локальные приоритеты по каждому критерию иерархии. В противном случае заполненные аналитиком данные весов противоречивы.

На основе уже известных данных о категориях и url-адресах можно представить декомпозицию в данном методе следующим образом:

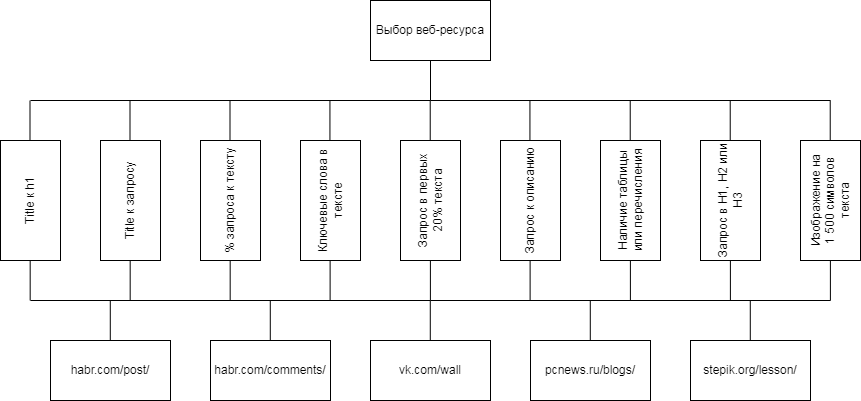


Рис. 3.11 – Иерархия проблемы выбора поставщика.

Таблица 3.1

Матрица оценки важности критериев

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Критерий | Title к h1 | Title к запросу | % запроса к тексту | Ключевые слова в тексте | Запрос в первых 20% текста | Запрос к описанию | Наличие таблицы или перечисления | Запрос в H1, H2 или H3 | Изображение на 1 500 символов текста | Произведение | Корень из произведения | Локальный вектор приоритетов |
| Title к h1 | 1 | 9 | 5 | 8 | 7 | 8 | 3 | 4 | 2 | 483 840 | 4,282 | 0,346 |
| Title к запросу | 1/9 | 1 | 1/5 | 1/3 | 3 | 7 | 2 | 1/2 | 8 | 1,25 | 1,025 | 0,083 |
| % запроса к тексту | 1/5 | 5 | 1 | 1/5 | 2 | 1/6 | 7 | 6 | 6 | 16,80 | 1,368 | 0,110 |
| Ключевые слова в тексте | 1/8 | 3 | 5 | 1 | 3 | 1/5 | 8 | 2 | 6 | 108,00 | 1,682 | 0,136 |
| Запрос в первых 20% текста | 1/7 | 1/3 | 1/2 | 1/3 | 1 | 1/4 | 6 | 1/3 | 4 | 0,016 | 0,631 | 0,051 |
| Запрос к описанию | 1/8 | 1/7 | 6 | 5 | 4 | 1 | 8 | 1/2 | 8 | 68,57 | 1,600 | 0,129 |
| Наличие таблицы или перечисления | 1/3 | 1/2 | 1/7 | 1/8 | 1/6 | 1/8 | 1 | 1/4 | 1/2 | 7,75 | 0,270 | 0,022 |
| Запрос в H1, H2 или H3 | 1/4 | 2 | 1/6 | 1/2 | 3 | 2 | 4 | 1 | 7 | 7,00 | 1,241 | 0,100 |
| Изображение на 1 500 символов текста | 1/2 | 1/8 | 1/6 | 1/6 | 1/4 | 1/8 | 2 | 1/7 | 1 | 1,55 | 0,292 | 0,024 |
| **Итого:** | | | | | | | | | | | **13,34** | **1,00** |

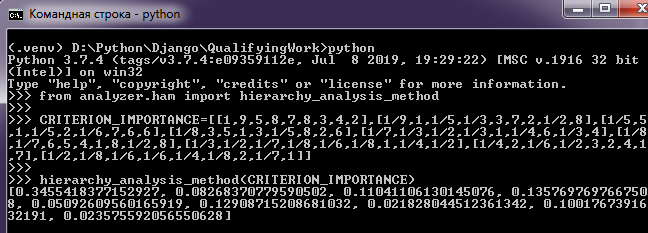


Рис. 3.12 – Результат нахождения локальных векторов

Дальше реализуем функцию определения приоритетных веб-страниц, предварительно реализовав функцию построения нахождения приоритетов критериев (Рис. 3.13).

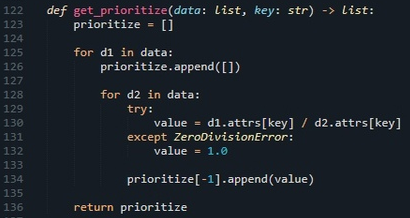


Рис. 3.13 – Функция нахождения приоритетов критериев.

В результате полученного кода были найдены приоритеты по всем анализируемым критериям (Рис. 3.14).

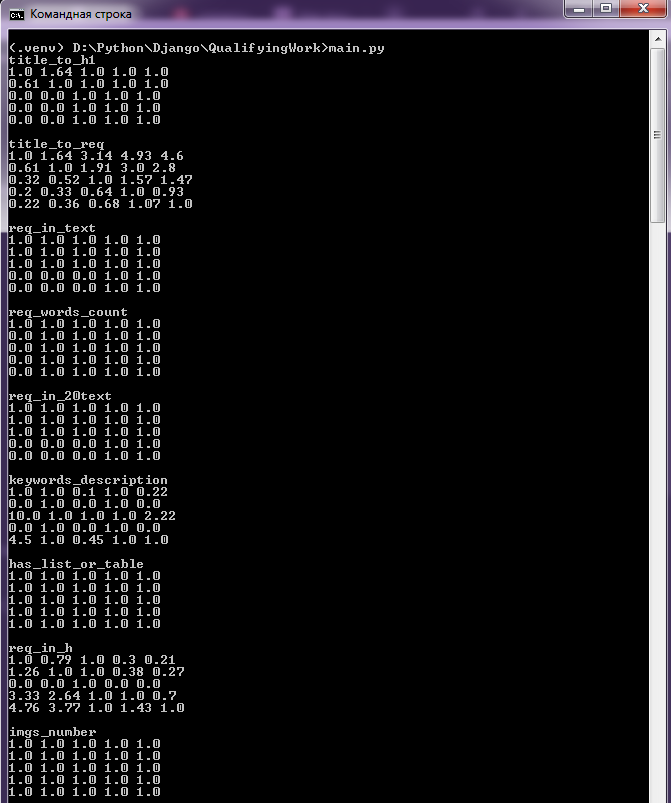


Рис. 3.14 – Приоритеты веб-страниц по показателям.

Чтобы определить, какому элементу предпочтительно отдать наибольший приоритет, необходимо составить матрицу глобального приоритета (Рис. 3.15).

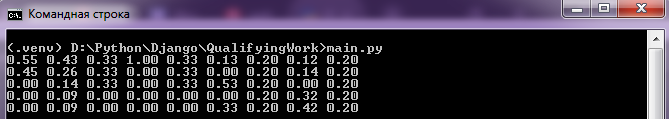


Рис. 3.14 – Пример вывода матрицы глобального приоритета.

Таким образом, можем определить глобальные приоритеты всех альтернатив иерархии, а затем, исходя из полученной информации, определить какая из них имеет больший вес над другими.

В результате выполнения алгоритма был выявлен веб-ресурс, наиболее отвечающий параметрам поисковой оптимизации (Рис. 3.15).

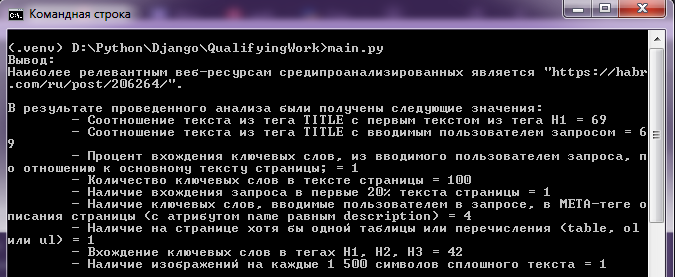


Рис. 3.15 – Вывод результата выполнения алгоритма оценки эффективности веб-ресурсов.

# **Заключение**

В данной выпускной работе был произведен анализ веб-страниц на предмет соответствия критериям поисковой оптимизации, а также их сравнение с результатами поисковой выдачи Google, построена математическая модель, анализирующая иерархию полученных критериев веб-ресурсов.

Для достижения поставленных задач были собраны и изучены показатели, характеризующие релевантность веб-ресурсов, построена, и реализована на языке программирования Python, модель сравнения двух множеств, а также реализован метод анализа иерархий.

Математические методы, реализованные в данной работе, выводят результаты о характеристике конкретных веб-страниц, что позволяет четко определить недостатки тех или иных проектных решений при верстке html кода для достижения более высоких позиций в списке выдачи поисковых систем.

Применение данной работы может быть полезно в области интернет-маркетинга. Аналитик, выявив необходимые критерии и определив степень их важности между друг другом, может провести сравнение собственных веб-страниц с конкурентными, чтобы определить вектор, в котором необходимо продвигать веб-страницу.

# **Список использованных источников**

1. https://www.reg.ru/blog/chto-takoe-seo-prodvizhenie-sajta-v-poiskoskovoj-sisteme/ - Поисковая оптимизация;

2. https://studopedia.net/14\_8407\_ponyatie-o-web-resurse-web-sayte-ih-klassifikatsiya.html - Веб-ресурсы;

3. https://habr.com/ru/post/232385/ - URI;

4. https://habr.com/ru/post/79210/ - Семантическая паутина;

5. https://ru.wikipedia.org/wiki/Расстояние\_Левенштейна - Расстояние Левенштейна;

6. В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965. 163.4:845-848;

7. https://edu.tltsu.ru/sites/sites\_content/site216/html/media67140/lec1\_is-2\_2020%20(1).pdf – Метод анализа иерархий (метод Т.Лю Саати);

8. https://studfile.net/preview/4634761/page:14/ - Случайная согласованность матриц;

9. https://requests.readthedocs.io/en/latest/ - Библиотека requests;

10. https://www.crummy.com/software/BeautifulSoup/bs4/doc/ - Библиотека bs4;

11. https://habr.com/ru/post/671136/ - Расстояние Хэмминга;

12. https://cyberleninka.ru/article/n/primenenie-matematicheskih-modeley-dlya-otsenki-effektivnosti-web-saytov/viewer - Применение математических моделей для оценки эффективности web-сайтов;

13. Программирование на Python. Том I. Четвертое издание - Марк Лутц;

14. Программирование на Python. Том II. Четвертое издание - Марк Лутц;

15. Курс высшей математики и математической физики под редакцией А.Н Тихонова, В.А.Ильина, А.Г. Свешникова. Выпуск 2. Основы математического анализа. Часть II;

16. Скрапинг веб-сайтов с помощью Python – Р. Митчелл;

17. Типизированный Python – А. Голобурдин;

18. Clean Code: A Handbook of Agile Software Craftsmanship – Robert C. Martin;

19. Python подробный справочник – Дэвид Бизли;

20. Python CookBook - Дэвид Бизли;

21. https://habr.com/ru/post/94707/ - RDF;

22/ http://book.uraic.ru/files/owl/REC-owl-guide-20040210\_ru.htm - OWL;

23. https://kartaslov.ru/значение-слова/семантическое+свойство – Семантическое свойство;

24. https://tproger.ru/translations/web-architecture-101/ - Архитектура веб-ресурсов;

25. https://habr.com/ru/post/480100/ - Виды оптимизации веб-ресурсов;

26. https://datareportal.com/ - Глобальная цифровая аналитика;

27. https://4brain.ru/blog/dekompoziciya-kak-instrument-dostizheniya-slozhnykh-celej/ - Декомпозиция;

28. https://vc.ru/seo/158794-kak-polzovateli-vedut-sebya-v-vydache-poiskovyh-sistem-issledovanie-dlya-google - «Как пользователи ведут себя в выдаче поисковых систем?»;

# **Приложение А**

***main.py:***

# -\*- coding: utf-8 -\*-

from parse.google\_api import get\_top\_pages

from analyzer import Analyzer

from req import Request

from analyzer.ham import hierarchy\_analysis\_method as HAM

from analyzer.ham import CRITERION\_IMPORTANCE

request\_text = 'Изобретаем JPEG'

# request\_text = input(

# 'Enter the query for which the analysis will be performed:'

# )

TRANSLATE\_INDICATOR\_NAME = {

'title\_to\_h1': 'Соотношение '

}

def main():

req = Request(request\_text)

pages = get\_top\_pages(req.text)

analyzers = [Analyzer(page, req) for page in pages]

priority\_web\_resource\_index = HAM(analyzers, CRITERION\_IMPORTANCE)

priority\_web\_resource = analyzers[priority\_web\_resource\_index]

print('Вывод:\nНаиболее релевантным веб-ресурсам среди'

f'проанализированных является "{priority\_web\_resource.page.url}".\n\n'

'В результате проведенного анализа были получены следующие значения:'

)

for key, value in priority\_web\_resource.items():

if \_\_name\_\_ == '\_\_main\_\_':

main()

***req.py:***

import string

def \_get\_keywords(text: str) -> tuple:

'''

Возвращает список слов из запроса, при этом удаляя знаки пунктуации

~params:

: text: str - текст запроса

'''

without\_punctuation = text.translate(

str.maketrans('', '', string.punctuation)

)

keywords = without\_punctuation.split()

return keywords

class Request:

'''

Представление запроса, по которому будет проходить анализ

~params:

: text: str - текст запроса

'''

def \_\_init\_\_(self, text: str):

self.text = text

self.words = \_get\_keywords(self.text)

def \_\_str\_\_(self):

return f'<Request "{self.text}">'

***parse/\_\_init\_\_.py:***

from .page import Page

\_\_all\_\_ = (

'Page',

)

***parse/page.py:***

from bs4 import BeautifulSoup

from .tools import get\_soup

from . import text as TextParser

from . import tags as TagParser

def \_get\_tag(soup: BeautifulSoup, tag\_name: str, text: bool=True) -> str:

tag = getattr(soup, tag\_name)

if text and tag is not None:

return tag.string

return ''

def get\_page\_information(url: str) -> dict:

'''

Собирает информацию о веб-странице.

~params:

: url: str - url-адрес.

'''

soup = get\_soup(url)

output = {

'title': \_get\_tag(soup, 'title'),

'h1': \_get\_tag(soup, 'h1'),

'ul': \_get\_tag(soup, 'ul', False),

'ol': \_get\_tag(soup, 'ol', False),

'table': \_get\_tag(soup, 'table', False),

'text': TextParser.get\_full\_text(soup),

'description': TextParser.get\_meta\_content(soup, 'description'),

'keywords': TextParser.get\_keywords(soup),

'captions': TagParser.get\_captions(soup),

'imgs': TagParser.get\_images(soup),

}

return output

class Page:

' Представление информации о веб-страницы '

def \_\_init\_\_(self, url: str):

self.url = url

page\_information = get\_page\_information(self.url)

self.\_\_dict\_\_.update(page\_information)

def \_\_str\_\_(self):

return f'<Page "{self.url}">'

***parse/tools.py:***

from urllib.parse import urlparse

import requests

from bs4 import BeautifulSoup

def get\_html(url: str, headers, \*\*params) -> str:

'''

Получает html код страницы

~params:

: url: str - url-адрес страницы;

: params - параметры строки запроса.

'''

response = requests.get(url, headers=headers, params=params)

html = response.text

return html

def get\_soup(url: str, headers: dict={}, \*\*params) -> BeautifulSoup:

'''

Создает экземпляр класса BeautifulSoup

~params:

: url: str - url-адрес страницы;

: params - параметры строки запроса.

'''

html = get\_html(url, headers, \*\*params)

soup = BeautifulSoup(html, 'html.parser')

return soup

***parse/text.py:***

from bs4 import BeautifulSoup

from bs4.element import Tag, Comment

INVISIBLE\_TEXT\_TAGS = (

'style',

'script',

'head',

'title',

'meta',

'[document]',

)

def is\_visible(text\_tag: Tag) -> bool:

'''

Возвращает 0, если тег с текстом является невидимым пользователю,

возвращает 1, если тег с текстом видим для пользователя.

~params:

: text\_tag: Tag - объект-тег

'''

if text\_tag.parent.name in INVISIBLE\_TEXT\_TAGS:

return False

if isinstance(text\_tag, Comment):

return False

return True

def get\_full\_text(soup: BeautifulSoup) -> str:

' Собирает весь видимый текст со страницы '

text\_tags = soup.find\_all(text=True)

visible\_text\_tags = filter(is\_visible, text\_tags)

full\_text = '\n'.join(t.strip() for t in visible\_text\_tags)

return full\_text

def get\_meta\_content(soup: BeautifulSoup, name: str) -> str:

meta\_tag = soup.find('meta', attrs={'name': name})

if meta\_tag is None:

return ''

return meta\_tag.attrs.get('content', '')

def get\_keywords(soup: BeautifulSoup) -> list:

keywords = get\_meta\_content(soup, 'keywords')

splited\_keywords = keywords.split(',')

keywords\_list = list(map(lambda s: s.strip(), splited\_keywords))

return keywords\_list

***parse/tags.py:***

from bs4 import BeautifulSoup

from bs4 import element

def get\_captions(soup: BeautifulSoup) -> tuple:

h\_tags = soup.find\_all('h1')

h\_tags.extend(soup.find\_all('h2'))

h\_tags.extend(soup.find\_all('h3'))

return tuple(h\_tags)

def \_is\_content\_img(img: element.Tag) -> bool:

href = img.attrs.get('href', '')

if 'logo' in href or 'icon' in href:

return False

return True

def get\_images(soup: BeautifulSoup) -> tuple:

all\_imgs = soup.find\_all('img')

filtered\_imgs = filter(\_is\_content\_img, all\_imgs)

return tuple(filtered\_imgs)

***parse/google\_api.py:***

from req import Request

from parse.tools import get\_soup

from parse import Page

\_BASE\_URL = 'https://www.google.com/search'

\_HEADERS = {

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 YaBrowser/22.11.5.715 Yowser/2.5 Safari/537.36',

'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9',

'accept-language': 'ru,en;q=0.9',

}

def get\_top\_page\_url(keywords: str) -> list:

'''

Возвращает url лучшего результата в Google поиске.

~params:

: keywords: str - текст с запросом

'''

soup = get\_soup(\_BASE\_URL, headers=\_HEADERS, q=keywords)

first\_blocks = soup.find\_all('div', class\_='yuRUbf')[:5]

hrefs = [first\_block.find('a').attrs.get('href', '') \

for first\_block in first\_blocks]

return hrefs

def get\_top\_page(keywords: str) -> Page:

page\_url = get\_top\_page\_url(keywords)[0]

page = Page(page\_url)

return page

def get\_top\_pages(keywords: str) -> list:

page\_urls = get\_top\_page\_url(keywords)

pages = [Page(page\_url) for page\_url in page\_urls]

return pages

***analyzer/\_\_init\_\_.py:***

from .analyzer import Analyzer

\_\_all\_\_ = (

'Analyzer',

)

***analyzer/analyzer.py:***

from parse import Page

from . import text as TextAnalyzer

from . import html as HTMLAnalyzer

from .ham import hierarchy\_analysis\_method

from req import Request

def get\_page\_analyze(page: Page, request: Request) -> dict:

output = {

'title\_to\_h1': TextAnalyzer.Correlation.text\_text(

page.title, page.h1),

'title\_to\_req': TextAnalyzer.Correlation.text\_text(

page.title, request.text),

'req\_in\_text': int(TextAnalyzer.text\_in\_text(

request.text, page.text)),

'req\_words\_count': TextAnalyzer.compare\_keywords\_count(

page, request),

'req\_in\_20text': int(TextAnalyzer.text\_in\_text(

request.text, TextAnalyzer.\_get\_text\_percent(page.text, 20))),

'keywords\_description': TextAnalyzer.Correlation.text\_list(

page.description, request.words),

'has\_list\_or\_table': int(HTMLAnalyzer.has\_tags(

page, 'ol', 'ul', 'table')),

'req\_in\_h': TextAnalyzer.request\_in\_captions(

request.text, page.captions),

'imgs\_number': int(HTMLAnalyzer.imgs\_number\_in\_text(

page.text, page.imgs)),

}

return output

class Analyzer:

'''

Представление анализа данных страницы.

~params:

: page: Page - разобранная информация о странице.

'''

def \_\_init\_\_(self, page: Page, request: Request):

self.page = page

self.request = request

self.attrs = get\_page\_analyze(self.page, self.request)

def \_\_str\_\_(self):

return f'<Analyzer "{self.page.url}">'

***analyzer/ham.py:***

# hierarchy analysis method

from typing import Union

def P(\*args) -> Union[float, int]:

output = args[0]

for number in args[1:]:

output \*= number

return output

# Важность критериев

CRITERION\_IMPORTANCE = [

[1, 9, 5, 8, 7, 8, 3, 4, 2 ],

[1/9, 1, 1/5, 1/3, 3, 7, 2, 1/2, 8 ],

[1/5, 5, 1, 1/5, 2, 1/6, 7, 6, 6 ],

[1/8, 3, 5, 1, 3, 1/5, 8, 2, 6 ],

[1/7, 1/3, 1/2, 1/3, 1, 1/4, 6, 1/3, 4 ],

[1/8, 1/7, 6, 5, 4, 1, 8, 1/2, 8 ],

[1/3, 1/2, 1/7, 1/8, 1/6, 1/8, 1, 1/4, 1/2],

[1/4, 2, 1/6, 1/2, 3, 2, 4, 1, 7 ],

[1/2, 1/8, 1/6, 1/6, 1/4, 1/8, 2, 1/7, 1 ],

]

# Случайная согласованность

RANDOM\_CONSISTENCY\_PARAMS = [0, 0, .58, .9, 1.12, 1.24, 1.32, 1.41, 1.45, 1.49]

def get\_consistency\_attribute(criterion\_importance: list,

local\_vectors: list) -> float:

'''

Расчитывает отношение согласованности

~params:

: criterion\_importance: list - критерии оценки;

: local\_vectors: list - локальные векторы матрицы

'''

n = len(criterion\_importance[0])

# Индекс согласованности (ИС)

I\_max = sum([sum(col)\*local\_vectors[i] \

for i, col in enumerate(zip(\*criterion\_importance))])

consistency\_index = (I\_max - n) / (n - 1)

# Случайная согласованность

random\_consistency = RANDOM\_CONSISTENCY\_PARAMS[n-1]

# Отношение согласованности

consistency\_attribute = consistency\_index / random\_consistency

return consistency\_attribute

def get\_criterion\_vectors(criterion\_importance: list) -> float:

'''

Расчитывает локальный векторы критериев.

~params:

: criterion\_importance: list - критерии оценки

'''

n = len(criterion\_importance[0])

# Произведения строк важности

criterion\_product = [P(\*row) for row in criterion\_importance]

# Корени от произведения строк важности

number\_root = [product\*\*(1/n) for product in criterion\_product]

# Локальные векторы

local\_vectors = [nr/sum(number\_root) for nr in number\_root]

return local\_vectors

def get\_priority(criterion\_vectors: list, vectors: list) -> Union[int, float]:

s = 0

for cv, v in zip(criterion\_vectors, vectors):

s += cv \* v

return s

def get\_prioritize(data: list, key: str) -> list:

prioritize = []

for d1 in data:

prioritize.append([])

for d2 in data:

try:

value = d1.attrs[key] / d2.attrs[key]

except ZeroDivisionError:

value = 1.0

prioritize[-1].append(value)

return prioritize

def hierarchy\_analysis\_method(data: list, criterion\_importance: list):

criterion\_vectors = get\_criterion\_vectors(criterion\_importance)

consistency\_attribute = get\_consistency\_attribute(

criterion\_importance, criterion\_vectors)

# Проверка согласованности

cc = consistency\_attribute <= .1

if not cc:

raise ValueError('Некорректные весы критериев!!!')

criterions = []

for key in data[0].attrs.keys():

prioritize = get\_prioritize(data, key)

vectors = get\_criterion\_vectors(prioritize)

criterions.append(vectors)

# Матрица для расчета глобальных приоритетов

m = list(zip(\*criterions))

# Определение глобальных приоритетов

global\_priorities = []

for row in m:

temp = 0

for ind, col in enumerate(row):

value = col \* criterion\_vectors[ind]

temp += value

global\_priorities.append(temp)

max\_priority = max(global\_priorities)

return global\_priorities.index(max\_priority)

***analyzer/html.py:***

from parse import Page

def has\_tags(page: Page, \*tag\_names) -> bool:

'''

Проверка наличия на странице конкретных тегов.

~params:

: page: Page - экземпляр класса Page;

: tag\_names - название тегов.

'''

return any([hasattr(page, tag\_name) for tag\_name in tag\_names])

def imgs\_number\_in\_text(text: str, imgs: tuple) -> bool:

'''

Проверяет распределение количества картинок на длину текста.

Оптимально - 1 картинка на каждые 1500 символов текста.

~params:

: text: str - текст страницы;

: imgs: tuple - картинки на странице.

'''

text\_length = len(text)

imgs\_number = len(imgs)

return text\_length / imgs\_number <= 1500

***analyzer/levenshtein.py:***

from functools import lru\_cache

def match\_percent(func):

def wrapper(word\_1: str, word\_2: str, acceptable\_percent: int=75):

changes = func(word\_1, word\_2)

word\_length = len(word\_1) if len(word\_1) > 0 else 1

percent = 100 - int(changes / word\_length \* 100)

if percent >= acceptable\_percent:

return 100

return percent

return wrapper

@match\_percent

def lev(word\_1: str, word\_2: str) -> int:

@lru\_cache(None)

def min\_dist(s1: int, s2: int) -> int:

if s1 == len(word\_1) or s2 == len(word\_2):

return len(word\_1) - s1 + len(word\_2) - s2

if word\_1[s1] == word\_2[s2]:

return min\_dist(s1+1, s2+1)

return 1 + min(

# insert

min\_dist(s1, s2+1),

# delete

min\_dist(s1+1, s2),

# replace

min\_dist(s1+1, s2+1),

)

return min\_dist(0, 0)

***analyzer/text.py:***

from typing import Union

from parse import Page

from parse.google\_api import get\_top\_page

from req import Request

from .levenshtein import lev

def \_mean(\*values: Union[int, float]) -> int:

' Возвращает среднее значение переданных аргументов '

try:

output = int(abs(sum(values)) / len(values))

except ZeroDivisionError:

output = 0

return output

def \_get\_text\_percent(text: str, percent: Union[int, float]=100) -> str:

' Обрезает текст до определенного процента '

text\_length = len(text)

finish\_index = int(text\_length / 100 \* percent)

sliced\_text = text[:finish\_index]

return sliced\_text

class Correlation:

''' Функционал для сравнения строк '''

@staticmethod

def text\_text(text\_1: str, text\_2: str) -> int:

' Процент сравнения строк '

if None in (text\_1, text\_2):

return 0

return lev(text\_1.lower(), text\_2.lower())

@staticmethod

def text\_list(text: str, words: list) -> int:

' Процент вхождения слова в список '

lower\_words = tuple(map(lambda s: s.lower(), words))

output = [lev(text.lower(), word) for word in words]

value = sum(output)

return value if value >= 0 else 0

def text\_in\_text(text\_1: str, text\_2: str) -> bool:

'''

Возвращает True, если text\_1 входит в text\_2. Иначе - False.

~params:

: text\_1: str - текст, который должен входить в text\_2

: text\_2: str - общий текст в котором ищется вхождение

'''

return text\_1.lower() in text\_2.lower()

def request\_in\_captions(text: str, captions: Union[list, tuple]) -> int:

'''

Возвращает среднее вхождение запроса в теги.

~params:

: text: str - запрос;

: captions: [list, tuple] - список/кортеж тегов.

'''

correlation\_percents = [Correlation.text\_text(text, caption.text) \

for caption in captions]

text\_inclusion = \_mean(\*correlation\_percents)

return text\_inclusion

def words\_in\_text(words: list, text: str) -> int:

'''

Возвращает количество ключевых слов в тексте

~params:

: words: list - список ключевых слов;

: text: str - текст на странице.

'''

counts = sum([text.lower().count(word.lower()) for word in words])

return counts

def compare\_keywords\_count(page: Page, request: Request) -> int:

top\_page = get\_top\_page(request.text)

other\_keywords\_count = words\_in\_text(request.words, top\_page.text)

this\_keywords\_count = words\_in\_text(request.words, page.text)

keywords\_percent = (this\_keywords\_count // other\_keywords\_count) \* 100

return keywords\_percent