МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

РОСТОВСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ

УНИВЕРСИТЕТ (РИНХ)

**Факультет Компьютерных технологий и информационной безопасности**

**Кафедра Фундаментальной и прикладной математики**

ДОПУСТИТЬ К ЗАЩИТЕ

И.о. зав. кафедрой к.э.н. Рутта Н.А.

*.*

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**на тему:**

**«****ПРИМЕНЕНИЕ МАТЕМАТИЧЕСКИХ МЕТОДОВ И МОДЕЛЕЙ ДЛЯ ОЦЕНКИ ЭФФЕКТИВНОСТИ WEB-РЕСУРСОВ»**

|  |  |  |
| --- | --- | --- |
| Выполнил  студент группы № 351-ПМИOZ | *подпись* | *Р.И. Цупко* |
| Направление (специальность) | *01.03.02 «Прикладная математика и информатика»* | |
| Направленность | *01.03.02.01 «Математическое и информационное обеспечение финансово экономической деятельности»* | |
| Руководитель выпускной квалификационной работы |  | *Н.А. Рутта*  *К.Э.Н..- И.о. зав. кафедрой* |
|  | *подпись* |  |

Ростов-на-Дону, 2023

Оглавление

[**Введение** 3](#_Toc126703968)

[**Глава 1. ВЕБ-РЕСУРСЫ** 5](#_Toc126703969)

[1.1. Общие понятия 5](#_Toc126703970)

[1.2. Архитектура веб-ресурсов 6](#_Toc126703971)

[1.3. Поисковая оптимизация 8](#_Toc126703972)

[**2. Математические методы и модели оценки эффективности** 13](#_Toc126703973)

[Математическая модель 13](#_Toc126703974)

[Расстояние Левенштейна 14](#_Toc126703975)

[**3. Практическая часть** 15](#_Toc126703976)

[3.1 Структура кода 16](#_Toc126703977)

[**Заключение** 18](#_Toc126703978)

[**Список использованных источников** 19](#_Toc126703979)

[**Приложение А** 20](#_Toc126703980)

[Приложение Б 31](#_Toc126703981)

# **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:** Microsoft Excel, Python.

**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. ВЕБ-РЕСУРСЫ**

## 1.1. Общие понятия

**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).

**URI** – это последовательность символов, которая идентифицирует веб-ресурс, а также позволяет интернет-протоколам облегчить взаимодействие между этими ресурсами. 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** 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) и Web Ontology Language (OWL). 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.

**Семантические свойства** или свойства значения – 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. 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.

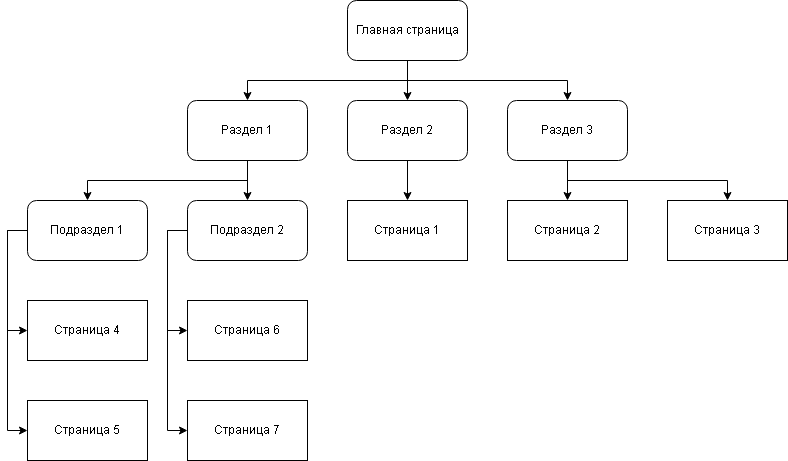


Рис. 1.1. Представление древовидной структуры веб-ресурса.

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. 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. 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.

# **2. Математические методы и модели оценки эффективности**

Математическая модель – это эффективное средство ответов, созданное людьми для решения проблем реального мира.

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.

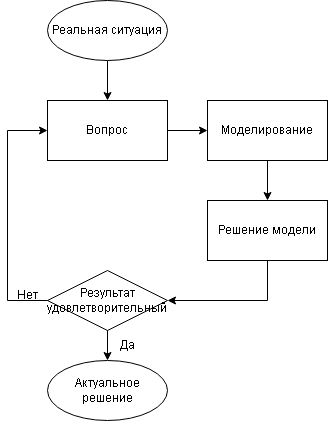


Рис. 2.1. Блок-схема иерархических требований.

## Расстояние Левенштейна

Для нахождения соотношений между двумя блоками текста будем использовать метрику для измерения разницы между двумя последовательностями – **расстояние Левенштейна**. 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 1.

Таблица 1

Первый шаг определения расстояния Левенштейна

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **-** | **К** | **Кр** | **Кри** | **Крик** |
| **-** | **0** | **1** |  |  |  |
| **Е** | **1** |  |  |  |  |
| **Ер** |  |  |  |  |  |
| **Ери** |  |  |  |  |  |
| **Ерик** |  |  |  |  |  |

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

Для символов «Кр» и «Ер» необходимо выполнить два действия удаления, следовательно, наша таблица примет вид (Таблица 2):

Таблица 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 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 4

Итоговый вариант 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).

# **3. Практическая часть**

Python - это высокоуровневый язык программирования общего назначения. Он использует динамическую типизацию и комбинацию подсчета ссылок, а также сборщик мусора для управления памятью. Отдельное внимание стоит уделить Python Zen [7], который определяет вектор развития языка и включает такие принципы как:

«Красивое лучше некрасивого»;

«Явное лучше неявного»;

«Простое лучше сложного»;

«Плоский лучше, чем вложенный»;

«Удобство чтения имеет значение»;

и т.д.

Также значительную роль играет возможность расширения возможностей средствами модулей, которые могут использоваться отдельными программами, не встраивая в ядро самого языка. Python отличает от множества популярных языков программирования фактор наличия динамической типизации, а также интерпретируемость, что в свою очередь позволяет исполнять команды в оболочке самого языка.

Интерпретируемые языки программирования – это такие языки, код которых выполняется построчно, а не компилируется в машинный. Однако есть и минус: интерпретируемые языки работают медленнее компилируемых за счет затрат на чтение строк, вместо единоразового преобразования в машинный код.

Сообщество Python создало большое количество модулей и фреймворков для облегчения решения множества проблем, на решение которых может уйти множество времени. В этой главе рассмотрим библиотеки и .

**requests** – это библиотека, предназначенная для отправки HTTP запросов (GET, POST, DELETE, PUT и др.);

**bs4** – предназначен для анализа, поиска и изъятия данных из html или lxml кода.

## 3.1 Структура кода

Для простоты понимания программный код был разбит на логические модули , отвечающий за получение информации с веб-страницы, и , анализирующий полученную информацию. Структура каталога приведена на Рис. 3.1.

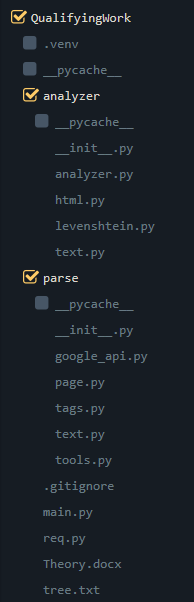


Рис. 3.1. Структура кода

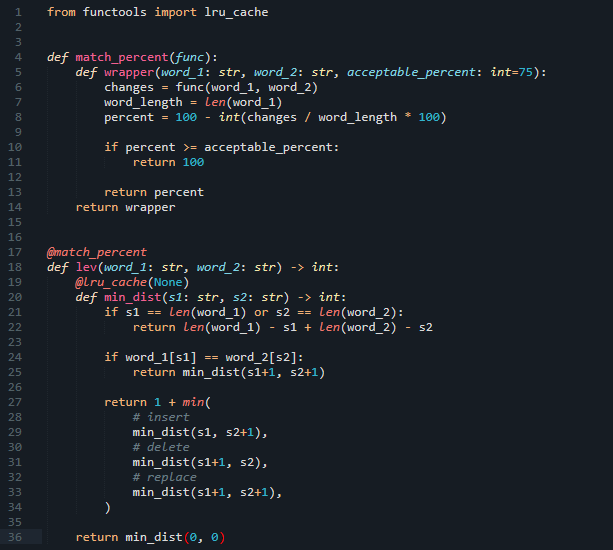


Рис. 2.2. Реализация метрики расстояния Левенштейна на языке программирования Python

# **Заключение**

# **Список использованных источников**

1. https://ru.wikipedia.org/wiki/Поисковая\_оптимизация - Поисковая оптимизация;

2. Фонталин Е. Оценка эффективности корпоративного сайта, 2014;

3. https://metrica.yandex.com/about - Яндекс.Метрика;

4. https://analytics.google.com/analytics/web - Google.Analytics;

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://ru.wikipedia.org/wiki/Дзен\_Пайтона - Python Zen;

# **Приложение А**

***main.py:***

\*Вставить код\*

***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/page.py:***

from .tools import get\_soup

from . import text as TextParser

from . import tags as TagParser

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

'''

Собирает информацию о веб-странице.

~params:

: url: str - url-адрес.

'''

soup = get\_soup(url)

output = {

'title': soup.title.string,

'h1': soup.h1.string,

'ul': soup.ul,

'ol': soup.ol,

'table': soup.table,

'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/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/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

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

from parse import Page

from . import text as TextAnalyzer

from . import html as HTMLAnalyzer

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

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

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

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

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

***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/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 '

return int(sum(values) / len(values))

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 '

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]

return output

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

***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)

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: str, s2: str) -> 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)

# Приложение Б