**МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**
**(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)**

**Институт №8 «Компьютерные науки и прикладная математика»**

# Лабораторные работы
# по курсу «Информационный поиск»

**Выполнил:** ***Ермаков Ярослав Валерьевич***

**Группа:** ***М8О-407Б-22***

**Преподаватели:** ***А.А. Кухтичев***

**Москва,** ***2025***

# Лабораторная работа №1

## Условия

Необходимо подготовить корпус документов, который будет использован при выполнении остальных лабораторных работ:

- Скачать его к себе на компьютер. В отчёте нужно указать источник данных.
- Ознакомиться с ним, изучить его характеристики. Из чего состоит текст? Есть ли дополнительная мета-информация? Если разметка текста, какая она?
- Разбить на документы.
- Выделить текст.
- Найти существующие поисковики, которые уже можно использовать для поиска по выбранному набору документов (встроенный поиск Википедии, поиск Google с использованием ограничений на URL или на сайт). Если такого поиска найти невозможно, то использовать корпус для выполнения лабораторных работ нельзя!
- Привести несколько примеров запросов к существующим поисковикам, указать недостатки в полученной поисковой выдаче.

В результатах работы должна быть указаны статистическая информация о корпусе:

- Размер «сырых» данных.
- Количество документов.
- Размер текста, выделенного из «сырых» данных.
- Средний размер документа, средний объём текста в документе.

## Описание

Целью работы является подготовка корпуса документов для последующего использования в лабораторных работах по информационному поиску. В ходе выполнения работы требуется выбрать не менее двух источников документов, загрузить примеры документов, изучить структуру данных, выделить текст из “сырого” формата и собрать статистические характеристики корпуса.

В качестве источников данных были выбраны два независимых открытых научных источника: **ACL Anthology** и **arXiv**. Эти источники подходят для задач поиска, так как содержат большое количество документов с устойчивыми URL, имеют доступ к HTML-страницам публикаций

- **ACL Anthology** - открытая библиотека публикаций по компьютерной лингвистике и смежным областям. Документы доступны по стабильным URL вида https://aclanthology.org/... и представлены в HTML с заголовком публикации и текстовым содержимым страницы.
- **arXiv** - открытый архив научных статей. Документы доступны по URL вида https://arxiv.org/abs/<id> и представлены HTML-страницами с метаинформацией и содержательными фрагментами.

По результатам выгрузки корпуса получены следующие статистические характеристики:

1. **Размер «сырых» данных:** 529 704 878 байт
2. **Количество документов:** 58 866.
3. **Размер текста, выделенного из «сырых» данных:** 529 704 878 байт. Выделение текста из HTML-разметки выполнялось на этапе подготовки корпуса, поэтому дальнейшие измерения проводились по уже очищенному текстовому представлению документов, используемому в последующих лабораторных работах.
4. **Средний размер документа:** 8998,49 байт.
5. **Средний объём текста в документе:** около 9 КБ очищенного текстового содержимого

## Исходный код

Для первой ЛР использовалась утилита export_corpus.py, предназначенная для подготовки корпуса документов к дальнейшим лабораторным работам. Программа подключается к MongoDB, выбирает последнюю сохранённую версию HTML-документа для каждого URL, извлекает из HTML видимый текст и заголовок страницы, нормализует пробелы и сохраняет очищенный текст в файлы docs/*.txt. Дополнительно формируется таблица meta.tsv с метаданными (URL, источник, время загрузки, заголовок, длина текста), а при необходимости сохраняется и “сырой” HTML в сжатом виде.

## Выводы

В ходе первой лабораторной работы был выбран и проанализирован корпус документов из двух открытых источников - ACL Anthology и arXiv. Были изучены структура исходных документов, наличие метаинформации и существующие средства поиска по данным источникам. Документы сохранены в исходном формате и подготовлены для последующей обработки и использования в дальнейших лабораторных работах по информационному поиску.

# Лабораторная работа №2

## Условия

Необходимо написать парсер на любом языке программирования.

- Написать поисковый робот - компоненты обкачки документов, используя любой язык программирования;
- Единственным аргументом поисковому роботу подаётся путь до yaml-конфига, содержащий:
  - Данные для базы данные в секции db;
  - Данные для робота в секции logic: задержка между обкачкой страницы;
  - Любые другие данные, необходимые для реализовации логики поискового робота.
- Сохранять в базе данных (например, MongoDB) документы со следующими полями:
  - url, нормализованный;
  - «сырой» html-текст документа;
  - название источника;
  - Дата обкачки документа в формате Unix time stamp.
- Поисковый робот можно остановить в любой момент и при повторном запуске робот должен начать с того документа, с которого он остановился;
- Периодически он должен уметь переобкачивать документы, которые уже есть в базе, но только в том случае, если они изменились.

## Описание

Целью второй лабораторной работы является разработка поискового робота для автоматической обкачки документов из открытых интернет-источников и сохранения их в базе данных для последующего использования в задачах информационного поиска. Поисковый робот реализует полный цикл загрузки документов: формирование очереди URL, выполнение HTTP-запросов, получение HTML-страниц и сохранение сырых данных вместе с метаинформацией. Управление параметрами работы робота осуществляется через YAML-конфигурационный файл, который передаётся программе в качестве единственного аргумента командной строки. В конфигурации задаются параметры подключения к MongoDB, а также основные параметры логики обхода, включая задержки между запросами и интервалы переобкачки документов.

## Журнал выполнения и решение проблем

В процессе выполнения лабораторной работы основное внимание было уделено обеспечению устойчивой и долговременной работы поискового робота. Так как обкачка документов может занимать продолжительное время, было важно обеспечить возможность безопасной остановки программы и её последующего запуска без потери прогресса. Для этого состояние обхода полностью хранится в базе данных: информация о URL, времени последней обкачки и запланированном времени следующей проверки сохраняется между запусками программы. Это позволяет при повторном запуске продолжить обработку документов, не начиная процесс заново.

Дополнительной задачей являлось предотвращение дублирования данных и избыточной переобкачки документов. Для её решения реализована логика повторной загрузки страниц только при изменении их содержимого. В ходе работы также учитывались сетевые ошибки и ограничения со стороны удалённых серверов, поэтому были введены задержки между запросами и механизм временной блокировки URL при параллельной обработке, что позволяет избежать конфликтов между потоками и обеспечивает корректное восстановление работы после аварийного завершения.

## Исходный код

Поисковый робот реализован на языке Python и использует стандартные средства работы с конфигурационными файлами и базами данных. В коде реализованы компоненты нормализации URL, выполнения HTTP-запросов, сохранения сырых HTML-документов и метаинформации, а также логика управления очередью URL. Архитектура программы ориентирована на хранение всего состояния в базе данных, что обеспечивает независимость работы робота от оперативной памяти и позволяет гибко управлять параметрами обхода через конфигурационный файл без изменения исходного кода.

## Выводы

В ходе выполнения второй лабораторной работы был разработан поисковый робот, обеспечивающий автоматическую обкачку документов, сохранение сырых HTML-данных и метаинформации в базе данных, а также возможность остановки и возобновления работы без потери прогресса. Реализованная система поддерживает переобкачку документов только при изменении их содержимого и удовлетворяет требованиям задания, создавая надёжную основу для формирования корпуса документов,

используемого в последующих лабораторных работах по информационному поиску.

# Лабораторная работа №3

## Условия

Нужно реализовать процесс разбиения текстов документов на токены, который потом будет использоваться при индексации. Для этого потребуется выработать правила, по которым текст делится на токены. Необходимо описать их в отчёте, указать достоинства и недостатки выбранного метода. Привести примеры токенов, которые были выделены неудачно, объяснить, как можно было бы поправить правила, чтобы исправить найденные проблемы.

В результатах выполнения работы нужно указать следующие статистические данные:

- Количество токенов.
- Среднюю длину токена.

Кроме того, нужно привести время выполнения программы, указать зависимость времени от объёма входных данных. Указать скорость токенизации в расчёте на килобайт входного текста. Является ли эта скорость оптимальной? Как её можно ускорить?

## Описание

Цель работы - реализовать токенизацию текстов документов корпуса, чтобы затем использовать полученные токены при построении индексов и поиске. Токенизация выполняется как последовательный проход по тексту документа с формированием токенов по заданным правилам, с приведением к единой форме и дополнительной нормализацией для русского языка. Отдельно предусмотрен режим со стеммингом, чтобы на следующих этапах уменьшать число различных словоформ.

## Правила токенизации

В качестве токена рассматривается непрерывная последовательность символов, относящихся к «словным» или «числовым» (буквы латиницы/кириллицы и цифры). Разделителями считаются пробельные символы и знаки пунктуации. Перед добавлением токена выполняются нормализации: понижение регистра, приведение некоторых вариантов написания к одному виду (например, ё -> е). Для технических терминов допускаются составные формы: дефис внутри токена сохраняется, что позволяет не разрушать записи вида state-of-the-art, 0-shot, x-ray, номера версий и похожие

обозначения. Токены короче минимального порога (например, 1 символ) отбрасываются, чтобы уменьшить шум от одиночных букв и случайных символов.

Достоинства подхода: простая и быстрая реализация, линейная сложность по длине текста, устойчивая работа на больших объёмах данных, единая нормализация регистра и русских вариантов написания.

Недостатки: часть “шумных” токенов всё равно остаётся, а также возможны спорные случаи с апострофами, сокращениями и HTML/служебными фрагментами.

## Примеры неудачных токенов и возможные улучшения

При токенизации научных HTML-страниц встречаются токены, которые не несут смысловой нагрузки для поиска:

- Чистые числа и «нулевые» группы: 00, 000 и т.п. Они часто возникают из-за разметки, ссылок, идентификаторов и версий. (Улучшение: отбрасывать токены, состоящие только из цифр (или понижать их вес на этапе ранжирования)).
- Слова с апострофом и сокращения: 0's и подобные формы. (Улучшение: определить правило для апострофов: либо оставлять внутри токена только для буквенных сокращений, либо заменять апостроф на разделитель.)
- Составные технические идентификаторы: 000-document, 0-8b-instruct. Для поиска по смыслу они полезны не всегда, но могут быть полезны для технических запросов. (Улучшение: оставить как есть (для совместимости с техтекстами) или вводить отдельную нормализацию для “модельных/версионных” токенов.)
- Артефакты HTML/кодировок в тексте: иногда в поле title/тексте встречаются искажённые последовательности символов из-за особенностей HTML или кодировки. (Улучшение: усилить очистку HTML до токенизации (удалять навигационные элементы, мусорные строки, проверять корректность декодирования UTF-8).)

## Результаты и статистические данные

Токенизация была выполнена на полном корпусе из 58 866 документов. Получены следующие измерения:

- **docs = 58866** - сколько документов из списка было обработано токенизатором.
- **total_bytes = 529704878** - суммарный объём входного текста всех документов (в байтах), который был прочитан и токенизирован.
- **token_count = 70584169** - общее число выделенных токенов по всему корпусу.
- **avg_token_len_chars = 6.00442** - средняя длина одного токена в символах (среднее по всем токенам).
- **time_sec = 6.38815** - время работы программы токенизации на всём корпусе (в секундах).
- **tokens_per_kb = 136.45** - плотность токенов: сколько токенов в среднем приходится на 1 килобайт входного текста (удобно для сравнения разных корпусов/настроек).
- **stemming = 1** - стемминг был включён при токенизации (если 0, то токены сохранялись без стемминга).

## Время выполнения, зависимость от объёма и скорость

Алгоритм токенизации выполняет один линейный проход по тексту каждого документа, поэтому время работы пропорционально общему объёму входных данных(т.е линейна). На полном корпусе объём входного текста составил около 517 тыс. KiB, а время выполнения - 6.39 сек, что даёт скорость порядка ~81 000 KiB/сек. В пересчёте на токены это примерно ~11 млн токенов/сек.

Такая скорость для однопроходной токенизации близка к практическому пределу, и дальнейшее ускорение обычно упирается в ввод-вывод и обработку Unicode. **Потенциальные способы ускорения:** чтение крупными буферами/mmap, уменьшение числа выделений памяти, упрощение проверок символов, распараллеливание по документам, а также оптимизация/отключение стемминга при измерении “чистой” токенизации.

## Выводы

В лабораторной работе реализована токенизация корпуса документов с едиными правилами выделения токенов и нормализацией. Получены статистические показатели (количество токенов, средняя длина токена, время работы и скорость), а также выявлены типичные примеры “шумных” токенов и направления для улучшения качества токенизации на смешанных HTML-данных.

# Лабораторная работа №4

## Условия

Добавить в созданную поисковую систему лемматизацию. В простейшем случае, это просто поиск без учёта словоформ. В более сложном случае, можно давать бонус большего размера за точное совпадение слов.

Лемматизацию можно добавлять на этапе индексации, можно на этапе выполнения поискового запроса. В отчёте должна быть включена оценка качества поиска, после внедрения лемматизации. Стало ли лучше? Изучите запросы, где качество ухудшилось. Объясните причину ухудшения и как можно было бы улучшить качество поиска по этим запросам, не ухудшая остальные запросы?

## Описание

Цель лабораторной работы - добавить в поисковую систему стемминг, чтобы поиск не зависел от словоформ и разные грамматические формы слова сопоставлялись как один терм. Стемминг применяется как нормализация токенов и позволяет повысить полноту поиска: документы находятся даже тогда, когда ключевое слово в тексте встречается в другой форме. Для корректной работы одинаковые правила обработки должны применяться как при индексации, так и при обработке поискового запроса.

## Журнал выполнения и решение проблем

Стемминг был внедрён как дополнительный шаг после токенизации: сначала выделяются токены по правилам и выполняется базовая нормализация , после чего при включённом режиме токен преобразуется стеммером. Основная практическая проблема при внедрении заключалась в том, что стемминг может не только улучшать выдачу, но и ухудшать её. Это проявляется в случаях, когда разные слова сводятся к одному стему, из-за чего в результатах появляются нерелевантные документы. Также на “технических” токенах стемминг может давать нестабильные эффекты, поэтому такие токены целесообразно либо не стеммировать, либо обрабатывать отдельным правилом. В ходе проверки также было важно обеспечить согласованность режимов: если индекс построен со стеммингом, то запрос должен обрабатываться тем же режимом, иначе результаты будут неполными.

## Исходный код

В системе реализован режим включения стемминга через параметр запуска (например,

--stemming 0/1). При построении индекса каждый токен перед добавлением в индекс проходит одинаковую цепочку преобразований, включая стемминг в соответствующем режиме. При выполнении запроса входная строка токенизируется теми же правилами и, при включённом режиме, токены также пропускаются через стеммер, после чего выполняется поиск по индексу. Такой подход делает поиск устойчивым к словоформам и обеспечивает корректное сопоставление запроса и данных индекса.

## Оценка качества поиска и анализ ухудшений

Оценка качества выполнялась сравнением выдачи для одинаковых запросов в двух режимах: без стемминга и со стеммингом. В большинстве случаев стемминг улучшает полноту: увеличивается количество релевантных документов, особенно для запросов с русскими словами, где в текстах часто встречаются разные падежи, числа и производные формы. При этом обнаруживаются запросы, где качество ухудшается. Основные причины ухудшения связаны с “слипанием” разных слов в один стем , из-за чего выдача расширяется за счёт нерелевантных документов. Также ухудшение может происходить для коротких токенов и для терминов с цифрами/дефисами, где преобразование не соответствует ожиданиям пользователя.

Чтобы улучшить качество на таких запросах, не ухудшая остальные, можно использовать комбинированный подход: выполнять поиск по стемам, но дополнительно повышать приоритет документов, где встречается точная форма слова из запроса, либо ограничивать применение стемминга для коротких токенов и технических обозначений. Ещё один практический вариант - хранить в индексе как стем, так и исходную форму и учитывать это на этапе ранжирования.

## Выводы

В лабораторной работе в поисковую систему был добавлен стемминг, применяемый согласованно при индексации и при обработке запросов. В результате поиск стал менее зависимым от словоформ, что повысило полноту выдачи. При этом выявлены запросы, где качество может снижаться из-за “слипания” разных слов в один стем и из-за особенностей технических токенов. Для снижения негативного эффекта целесообразно ограничивать стемминг для отдельных классов токенов и учитывать точные совпадения при формировании выдачи.

# Лабораторная работа №5

## Условия

Для своего корпуса необходимо построить график распределения терминов по частотностям в логарифмической шкале, наложить на этот график закон Ципфа. Объяснить причины расхождения.

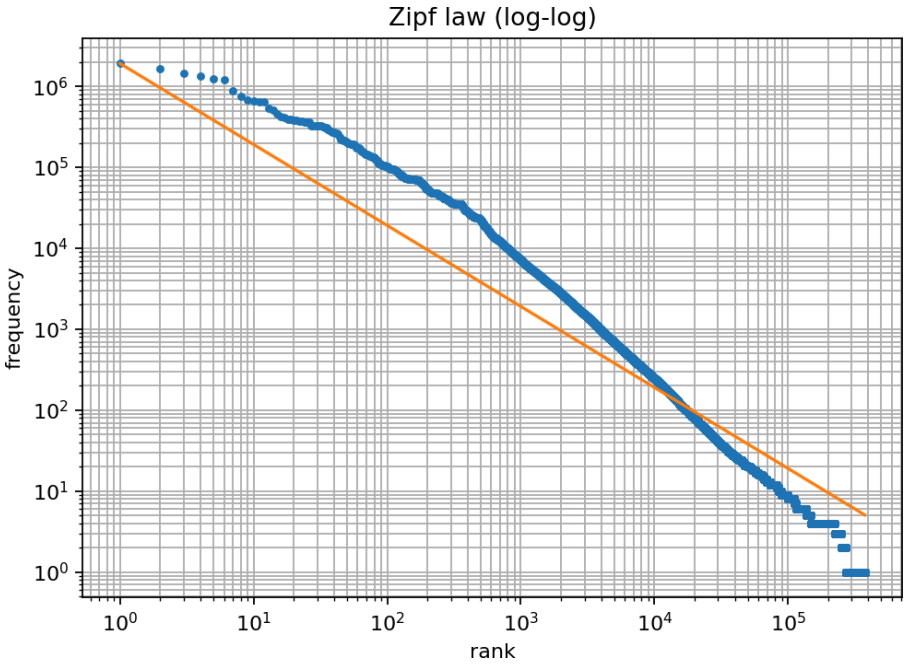
В качестве дополнительного задания, можно (но необязательно) подобрать константы для закона Мандельброта, наложить полученный график на график распределения терминов по частотностям. Привести выбранные константы.

## Описание

Целью лабораторной работы является проверка выполнения закона Ципфа на собранном корпусе. Для этого строится распределение терминов по частотам: термы сортируются по убыванию встречаемости, каждому терму присваивается ранг, после чего строится график зависимости **частоты от ранга** в **логарифмической шкале**. На полученный график накладывается теоретическая кривая по закону Ципфа, чтобы оценить соответствие реального распределения модели

## Журнал выполнения и решение проблем

Для построения распределения по частотам сначала были посчитаны частоты терминов по всему корпусу после токенизации и нормализации (в том числе со стеммингом). Для получения графика далее выполнялась сортировка терминов по убыванию частоты и построение набора точек вида (rank, freq). Чтобы график был информативным на широком диапазоне частот, обе оси были переведены в логарифмическую шкалу. На практике основной сложностью является наличие “шумных” терминов (числовые последовательности, версии, технические идентификаторы, артефакты HTML), которые заметно влияют на хвост распределения и усиливают расхождение с идеальным законом.




**ТОП-10 терминов(term-count):**

1. the 1914451
2. and 1654312
3. type 1465962
4. namepart 1341016
5. to 1245211
6. of 1204498
7. for 891166
8. is 759947
9. in 683674
10. role 656863

## Исходный код

В рамках работы использовались утилиты конвейера корпуса: токенизация и подсчёт частот терминов по документам, затем формирование файла распределения для построения графика. Построение графика выполнялось по данным “терм-частота” с последующей сортировкой и преобразованием в “ранг-частота”, после чего поверх эмпирических точек накладывалась теоретическая кривая Ципфа.

## Результаты и объяснение расхождения с законом Ципфа

График распределения терминов по частотам в логарифмической шкале демонстрирует характерную для текстов картину: небольшое количество самых частых терминов образует “голову” распределения, затем следует почти линейный участок в log-log координатах, и далее - длинный “хвост” редких терминов. Для наложения закона Ципфа использовалась зависимость вида $f(r) = C / r$, где r - ранг терма, а C выбиралась как частота самого частотного терма (то есть так, чтобы кривая начиналась с первой точки распределения).

Расхождения с идеальным законом Ципфа объясняются следующими факторами: корпус является смесью разных источников и стилей текста, поэтому распределение формируется не одним “однородным” языковым процессом; в данных присутствуют шумовые токены (числа, идентификаторы, версии, служебные фрагменты HTML), которые увеличивают долю редких терминов и “утяжеляют хвост”; кроме того, стемминг и правила токенизации объединяют часть словоформ и изменяют частоты некоторых групп терминов. Также влияние оказывает конечный размер корпуса: в реальных данных многие термы встречаются 1-2 раза, что создаёт плотный хвост и отклонение от гладкой теоретической кривой.

## Выводы

В ходе лабораторной работы было построено распределение терминов по частотам в логарифмической шкале и выполнено сравнение с законом Ципфа. Распределение в целом соответствует ожидаемой форме (почти линейное поведение в log-log координатах на среднем диапазоне рангов), а основные расхождения объясняются неоднородностью корпуса, наличием шумовых токенов и влиянием предобработки текста (токенизация и стемминг), а также большим количеством редких терминов в хвосте распределения.

# Лабораторная работа №7

## Описание

Целью работы является построение обратного индекса, пригодного для булева поиска, по подготовленному корпусу документов. Индекс строится в собственном бинарном формате и предназначен для дальнейшего расширения в следующих лабораторных работах. Помимо обратного индекса формируется прямой индекс, содержащий ссылки на документы и их заголовки, чтобы по идентификаторам документов можно было восстанавливать человекочитаемую выдачу. В качестве термов используются токены после нормализации и, при включённом режиме, после стемминга.

## Журнал выполнения и решение проблем

При построении индекса основная сложность заключалась в ограничениях на структуры данных: нельзя использовать ассоциативные контейнеры, поэтому нельзя напрямую “копить” словарь термов в map/unordered_map. Для решения использована внешняя сортировка: из документов формируются пары “терм-doc_id”, далее они сортируются по частям и сливаются в итоговый индекс. На практике возникла проблема согласования идентификаторов документов: метаданные из выгрузки содержали строковый идентификатор, а индексу требовались плотные числовые doc_id. Это было решено приведением метаданных к формату, где doc_id соответствует номеру документа в списке корпуса. После этого индекс успешно строится на полном наборе документов.

## Исходный код

Индексатор реализован на C++ и формирует три файла: прямой индекс docs.bin, словарь terms.bin и постинги postings.bin. В docs.bin хранятся URL и заголовки документов, в terms.bin - список термов и смещения на соответствующие постинги, а postings.bin содержит последовательности doc_id для каждого терма. Формат бинарный и расширяемый за счёт заголовков с magic и version. Для построения используется внешняя сортировка: данные разбиваются на несколько “run”-файлов, каждый run сортируется, затем выполняется k-way merge, в ходе которого формируются словарь и постинги.

## Выводы

В ходе работы построен булев индекс по корпусу из 58866 документов. В процессе внешней сортировки было создано 10 промежуточных run-файлов, итоговое число уникальных термов составило 376586, а размер файла постингов

- 79223688 байт. Полученный индекс соответствует требованиям: использован собственный бинарный формат, создан прямой индекс для выдачи, обеспечена возможность расширения формата в будущих лабораторных работах.

# Лабораторная работа №8

## Описание

Целью работы является реализация булевого поиска по построенному обратному индексу. Поисковая система принимает запрос с логическими операторами и возвращает документы, удовлетворяющие булевому выражению. Для восстановления результата пользователю используется прямой индекс: по doc_id выводятся URL и заголовок документа.

## Журнал выполнения и решение проблем

Основной задачей было обеспечить корректную обработку логических операторов и скобок, а также согласованность обработки запроса и индекса. Для этого запрос токенизируется теми же правилами нормализации, что применялись при индексации. В ходе тестирования было учтено, что в оболочке bash символ ! интерпретируется как служебный, поэтому запросы с отрицанием корректно запускать в кавычках или экранировать !. После настройки формы запуска запросов и согласования режима обработки булев поиск стал возвращать корректные результаты.

## Исходный код

Поисковый модуль реализован как CLI-утилита, которая загружает бинарные файлы индекса, разбирает запрос в булево выражение и вычисляет результат через операции над отсортированными списками doc_id. Для AND выполняется пересечение списков, для OR

- объединение, для NOT - разность с универсальным множеством документов. Скобки поддерживаются за счёт преобразования выражения в постфиксную форму и последующего вычисления по стеку.

## Выводы

Реализован булев поиск по собственному бинарному индексу. Поддерживаются операторы AND/OR/NOT (а также их символьные формы & | !) и скобки, выдача формируется через прямой индекс и содержит ссылки и заголовки документов.
Реализация корректно работает на полном корпусе и является базой для последующих улучшений поиска и ранжирования