Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Отчет по лабораторным работам №по курсу «Информационный поиск»

Студент: А. О. Ларченко
Преподаватель: А. А. Кухтичев
Группа: М8О-406Б-22

Дата: 01.02.2026
Оценка:
Подпись:

Москва, 2026

Содержание

1 Добыча корпуса документов

Цель этапа — сформировать корпус документов для дальнейшей работы.

Задачи:

- получить список документов (URL) по заданной тематике и сохранить их вместе с метаданными;
- обеспечить воспроизводимость и повторяемость процесса (запуск из Docker, сохранение результатов в томах).

Выбранная тема - Искусство.

1.1 Параметры конфигурации

Пайплайн настраивается двумя файлами: `.env` (переменные окружения) и `config.yaml` (логика и параметры добычи).

Переменные окружения (`.env`).

- `MYSQL_HOST`, `MYSQL_PORT`, `MYSQL_USER`, `MYSQL_PASSWORD` — параметры подключения bootstrap-контейнера к MySQL.
- `UID`, `GID` — запуск контейнеров от имени текущего пользователя (чтобы файлы в `./data` не создавались с владельцем `root`).

Ключевые разделы `config.yaml`.

- `sources` — список источников: `ruwiki` (`ru.wikipedia.org`) и `ruwikisource` (`ru.wikisource.o`
- `bootstrap` — параметры формирования seeds:
  - `wiki_projects` — проекты Wikimedia, для которых скачиваются дампы и создаются базы в MySQL;
  - `dump_resources` — таблицы, которые нужны для навигации по категориям и страницам (`page`, `categorylinks`, `linktarget`);
  - `seed_category` — стартовая категория («Искусство»);
  - `max_depth` — максимальная глубина обхода графа категорий;
  - `seed_limit` / `logic.max_unique_docs` — целевой размер итогового списка документов;

– `timeout_sec.connect`, `timeout_sec.read` — таймауты загрузки дампов.

- `seeds` — параметры выходного файла seeds (TSV), а также ограничения на число документов на источник.
- `normalization` — нормализация URL.

1.2 Исходные данные

В качестве источника данных используются проекты Wikimedia:

- **ruwiki** — русская Википедия;
- **ruwikisource** — русский Викитека/Викиисточник.

Для bootstrap-этапа используются **официальные SQL-дампы** в формате `.sql.gz`, которые скачиваются из репозитория Wikimedia Dumps по шаблону:

```
https://dumps.wikimedia.org/{project}/latest/{project}-latest-{resource}.sql.gz
```

В работе требуются три таблицы:

- `page` — список страниц (идентификатор, namespace, заголовок, признак редиректа и др.);
- `categorylinks` — связи “категория → элементы” (страницы и подкатегории);
- `linktarget` — справочник целей ссылок (используется для поиска категории по названию и namespace).

1.3 Алгоритм и шаги

Процесс добычи корпуса реализован в виде последовательных шагов, выполняемых в Docker Compose. Управление шагами производится командами Makefile.

1.3.1 Шаг 1. Подготовка MySQL и загрузка дампов (bootstrap)

Запуск:

```
make bootstrap-init
```

Внутри bootstrap выполняется скрипт `init.py`, который реализует следующие действия:

1. **Загрузка конфигурации:** путь к `config.yaml` передаётся через `CONFIG_PATH`.
2. **Проверка доступности MySQL:** выполняются повторные попытки подключения (`bd_attempts_count` раз с паузой `bd_attempts_dly` секунд).
3. **Создание баз данных:** для каждого проекта из `bootstrap.wiki_projects` создаётся отдельная база MySQL (например, `ruwiki`, `ruwikisource`).
4. **Скачивание дампов:** для каждой пары (проект, ресурс) формируется имя файла вида `{project}-latest-{resource}.sql.gz` и выполняется загрузка в каталог `/data/dumps/{project}/`.
5. **Импорт дампов:** содержимое `.sql.gz` распаковывается "на лету" и подаётся в `mysql` CLI на соответствующую базу данных.

   Для повторяемости реализованы **маркеры импорта**: после успешного импорта создаётся файл `/data/logs/import_markers/{project}__{resource}.json`, содержащий сигнатуру дампа (размер и время модификации). Если сигнатура не изменилась, повторный импорт пропускается.
6. **Проверка таблиц:** выполняется контроль наличия ожидаемых таблиц и печать их структуры (оценка числа строк, объёмы данных/индексов, список колонок).
7. **Печать примеров строк:** для основных таблиц выводится несколько строк (LIMIT 3) для ручной верификации корректности импорта.

Файлы дампов и логов сохраняются в примонтированную директорию проекта (на хосте):

- `./data/dumps/` — загруженные `.sql.gz`;
- `./data/logs/` — логи и маркеры импорта.

1.3.2 Шаг 2. Формирование списка seeds (выбор документов по тематике)

После импорта дампов bootstrap формирует список URL документов (seeds) в TSV-файл (по умолчанию: `/data/exports/seeds_all_Искусство.tsv`).
Логика формирования seeds:

1. В таблице `linktarget` находится идентификатор стартовой категории `seed_category` в namespace 14 (категории).
2. Выполняется обход графа категорий **в ширину** (BFS) до глубины `max_depth` по таблице `categorylinks`.

3. Для каждой посещённой категории собираются:
   - страницы (`cl_type = page`);
   - подкатегории (`cl_type = subcat`) для дальнейшего обхода.
4. По `page_id` запрашиваются поля из таблицы `page`; далее применяется фильтрация:
   - только основной namespace (`page_namespace = 0`), если включён флаг "только статьи";
   - исключение редиректов (`page_is_redirect = 1`);
   - удаление повторов по `page_id`.
5. Для каждой выбранной страницы строится URL вида `https://<domain>/wiki/<title>`, где пробелы в заголовке заменяются на подчёркивания.
6. Итоговый список формируется **из двух источников** (первые два источника из `config.sources`). При включённом `seeds.shuffle` кандидаты перемешиваются с фиксированным seed для воспроизводимости.
7. Размер итогового списка ограничивается целевым значением `logic.max_unique_docs`, при этом действует ограничение `seeds.max_per_source` на долю одного источника.

Файл seeds сохраняется в `./data/exports/` (на хосте) и имеет формат TSV со столбцами:

`url` `title` `page_id` `namespace` `source`

1.4 Результаты

По итогам выполнения этапа получены следующие артефакты корпуса:

- **Сырые данные (дампы)**: `./data/dumps/<project>/<project>-latest-<resource>.sql.g`
- **Логи и маркеры импорта**: `./data/logs/import_markers/<project>__<resource>.json`.
- **Список документов корпуса (seeds)**: `./data/exports/seeds_all_Искусство.tsv`.

Список документов корпуса будет использован дальше поисковым роботом для загрузки HTML-страниц и сохранения их в MongoDB.

2 Поисковый робот

Цель этапа — по списку документов (seeds), сформированному на этапе bootstrap, загрузить HTML-страницы и сохранить их в MongoDB для дальнейшей индексации и поиска.

Задачи поискового робота:

- прочитать файл seeds в формате TSV и сформировать очередь URL для обработки;
- выполнить загрузку HTML по каждому URL с учётом ограничений по доменам источников;
- обеспечить устойчивость к сетевым сбоям и ограничениям по частоте запросов (таймауты, ретраи, 429 Retry-After);
- сохранять результат в MongoDB с метаданными и поддержкой режима возобновления (checkpoint).

2.1 Параметры конфигурации

Поисковый робот настраивается через `.env` (значения переменных окружения) и `config.yaml` (логика краулинга и параметры HTTP).

Переменные окружения (`.env`).

- `MONGO_URI` — строка подключения к MongoDB.
- `MONGO_DB` — имя базы данных MongoDB.
- `MONGO_COLLECTION` — коллекция для **сырых** документов (HTML).
- `MONGO_COLLECTION_CLEAN` — коллекция для **очищенных** документов (результат работы `html_cleaner`).
- `DEBUG_LIMIT` — отладочное ограничение количества обрабатываемых seeds (если задано).

Ключевые разделы `config.yaml`.

- `db` — описывает, какие переменные окружения использовать для подключения к MongoDB: `uri_env`, `database_env`, `collection_env`, `clean_collection_env`.

- `seeds` — входной TSV-файл со списком документов: `file`, `has_header`, `shuffle`, `max_per_source`.
- `sources` — источники и разрешённые домены (поисковый робот жёстко фильтрует URL по домену).
- `logic` — параметры параллелизма и режима возобновления: `concurrency`, `per_host_concurre` `delay_sec`, `user_agent`, `max_unique_docs`, `resume`.
- `logic.checkpoint` — файл состояния и частота сохранения: `state_file`, `flush_every_n`.
- `logic.recrawl` — режим повторного обхода и проверки изменений: `enabled`, `revisit_after_hours`, `only_if_changed`, а также `change_detection` (`use_etag`, `use_last_modified`, `use_content_hash`).
- `http` — параметры загрузки: таймауты (`timeout_sec.connect/read`), ретраи (`retries.count/backoff_sec`), `max_redirects`, лимит размера ответа (`max_response_mb`).

2.2 Запуск поискового робота

Поисковый робот запускается отдельной make-целью:

```
make crawl-init
```

Команда выполняет следующие действия:

1. Поднимает MongoDB (`make up-mongo` как зависимость).
2. Собирает образ сервиса робота (`docker-compose build crawler`).
3. Запускает робота одноразово (`docker-compose run -rm crawler`).

2.3 Алгоритм работы

Алгоритм реализован в файле `crawler.py` и включает следующие шаги.

2.3.1 Шаг 1. Подключение к MongoDB и подготовка индексов

Поисковый робот читает значения переменных окружения, подключается к MongoDB и создаёт индексы:

- уникальный индекс по `url_norm` (защита от дублей);
- индекс по `source_name`;
- индекс по `fetched_at_unix`.

2.3.2 Шаг 2. Загрузка seeds и ограничение объёма обработки

Файл seeds читается как TSV (при необходимости пропускается заголовок `has_header`). Внутри робота используется минимально необходимая информация: `url` и `source`.

Далее применяются ограничения:

- если задана переменная `DEBUG_LIMIT`, берётся только первые N записей;
- дополнительно применяется лимит `logic.max_unique_docs`.

2.3.3 Шаг 3. Режим возобновления (checkpoint) и детерминированное перемешивание

Если включён `logic.resume`, поисковый робот работает с файлом состояния `logic.checkpoint.sta`
Файл хранит:

- `next_idx` — индекс следующего seeds для обработки;
- счётчики (`processed`, `ok`, `unchanged`, `failed`);
- `by_source` — статистику по источникам;
- `shuffle_seed` — фиксированный seed для воспроизводимого shuffle.

Если включён `seeds.shuffle`, то список seeds перемешивается **детерминированно** (через сохранённый `shuffle_seed`), чтобы при возобновлении порядок не «прыгал».

2.3.4 Шаг 4. Ограничения по параллелизму и задержки

Для устойчивости и корректного поведения по отношению к источникам применяется ограничитель:

- общий параллелизм: `logic.concurrency`;
- параллелизм на один домен: `logic.per_host_concurrency`;
- задержка между запросами на хост: `logic.delay_sec`.

2.3.5 Шаг 5. Загрузка HTML и обработка ответов

Загрузка выполняется через `aiohttp` и поддерживает:

- таймаут подключения и чтения (`http.timeout_sec`);

- повторные попытки с backoff (`http.retries`);
- ограничение на размер ответа (потоковое чтение до `http.max_response_mb`).

Перед загрузкой URL нормализуется (`normalization`): удаление фрагмента `#...`, приведение хоста к нижнему регистру, удаление завершающего слеша и т.д. Также проверяется, что итоговый домен входит в `sources.allowed_domains` для данного источника.

Повторный обход и проверка изменений (recrawl). Если документ уже есть в MongoDB и включён `logic.recrawl.enabled`, робот:

- проверяет, прошло ли достаточно времени с момента последней загрузки (`revisit_after_hou`
- при режиме `only_if_changed` добавляет условные заголовки `If-None-Match` (ETag) и/или `If-Modified-Since`.

Обработка статусов:

- `200 OK`: HTML сохраняется в MongoDB.
- `304 Not Modified`: документ считается неизменённым, обновляется только `fetched_at_unix`.
- `429 Too Many Requests`: ошибка считается **временной** (retryable). Робот читает `Retry-After` и ставит "cooldown" на хост, затем повторяет попытку.

2.3.6 Шаг 6. Сохранение документа в MongoDB

При успешной загрузке документ сохраняется с upsert по ключу `url_norm`. Сохраняемые поля:

- `url`, `url_norm`;
- `html` (строка, декодирование UTF-8 с заменой ошибок);
- `source_name`;
- `fetched_at_unix`;
- при наличии: `etag`, `last_modified`;
- при включённом контроле: `content_hash` (SHA-256 от тела ответа).

2.4 Очистка HTML (html_cleaner)

После загрузки HTML поисковым роботом выполняется очистка страницы до текста для индексации. Очистка реализована в файле `html_cleaner.py` и запускается командой:

```
make clean_html
```

2.4.1 Алгоритм очистки

Для каждого документа из коллекции `MONGO_COLLECTION`:

1. Извлекается заголовок страницы: `#firstHeading` (приоритетно), затем `<h1>`, затем `<title>`.
2. Определяется область основного контента: `#mw-content-text` и вложенные блоки `.mw-parser-output` (или аналоги).
3. Удаляются служебные элементы: `script/style/noscript`, оглавление, блоки редактирования, ссылки на источники, навигационные боксы, категории, метаданные и т.п.
4. Отсекаются финальные разделы Википедии по заголовкам `h2` (например: “Примечания”, “Ссылки”, “Литература”, “См. также”) вместе со всем содержимым ниже.
5. Формируется чистый текст через `get_text`, нормализуются пробелы, удаляются маркеры сносок вида `[1]`.

Результат сохраняется в коллекцию `MONGO_COLLECTION_CLEAN` в виде:

- `url_norm`, `url_clean`;
- `title`;
- `clean_text`;
- `src_id` — ссылка на исходный документ из “сырой” коллекции.

Для коллекции очищенных документов создаются индексы:

- уникальный индекс по `url_norm`;
- индекс по `source_name`;
- уникальный индекс по `src_id`.

2.5 Результаты

По итогам работы поискового робота и очистки HTML получены следующие артефакты:

- **Корпус HTML-документов (MongoDB)**: `MONGO_DB.MONGO_COLLECTION`.
- **Очищенный корпус (MongoDB)**: `MONGO_DB.MONGO_COLLECTION_CLEAN` с полями `title` и `clean_text`.
- **Файл состояния (checkpoint)**: `logic.checkpoint.state_file` для возобновления обработки.
- **Логи ошибок** (при наличии): `/data/logs/crawler_errors.log`.

3 Токенизация

3.1 Теоретические сведения

Токенизация (tokenization) — это преобразование исходного текста в последовательность *токенов* (слов/чисел/символических единиц), которые далее используются в задачах информационного поиска: построение индекса, подсчёт частот, поиск по терминам и т.д. Корректная токенизация критична, потому что определяет, *что именно* считается словом, какие символы отбрасываются, как обрабатываются регистр и кодировки.

Для русскоязычных коллекций важна поддержка UTF-8 и нормализация регистра, поскольку один и тот же термин может встречаться в разных вариантах: “Искусство”, “искусство”, “ИСКУССТВО”.

3.2 Требования и входные данные

На вход модулю токенизации подаётся строка текста (в работе — поле `clean_text` из коллекции очищенных документов MongoDB). Текст уже очищен от HTML-разметки и содержит преимущественно естественный язык.

Требования к токенам в реализованной версии:

- токен формируется из последовательности букв (латиница ASCII) и букв кириллицы (UTF-8), допускаются также цифры;
- все буквы приводятся к нижнему регистру;
- токены длиной меньше заданного порога отбрасываются (`MIN_LETTERS_CNT`);
- любые символы-разделители (пробелы, пунктуация и т.п.) завершают токен.

3.3 Реализация и структура программы

Токенизация реализована в файлах:

- `tokenizer.hpp/.cpp` — основной алгоритм токенизации и структура токена;
- `utf8_utils.hpp/.cpp` — утилиты для определения длины UTF-8 символа и проверки кириллицы.

Основной интерфейс — функция:

```
std::vector<std::string>tokenize(std::string s);
```

Внутри используется структура `Token`, которая накапливает текущий токен:

- `text` — текст токена;
- `start/end` — позиции в исходной строке (для отладки и расширения функциональности);
- `letters_cnt` — количество *символов* в токене (для UTF-8 важно, что байты и символы различаются).

3.4 Алгоритм токенизации

Алгоритм основан на линейном проходе по строке слева направо с учётом длины UTF-8 символов:

1. Инициализируется пустой текущий токен.
2. Для каждой позиции `i` берётся байт `s[i]` и вычисляется предполагаемая длина UTF-8 символа `letter_len` (1–4 байта) с помощью `utf8_len()`.
3. Если `letter_len == 1`, символ обрабатывается как ASCII:
   - если это латинская буква или цифра — добавляется в токен (буквы приводятся к нижнему регистру);
   - иначе считается разделителем и токен завершается.
4. Если `letter_len > 1`, сначала выполняется проверка корректности UTF-8 последовательности (`is_valid_utf8`):
   - если последовательность некорректна — текущий токен завершается, а символ считается разделителем;
   - если это 2-байтовая кириллица — символ добавляется в токен (с приведением регистра);
   - иначе (например, emoji, латиница вне ASCII, 3–4 байта) — символ считается разделителем.
5. При завершении токена он добавляется в выходной список, если длина токена по символам $\geq$ `MIN_LETTERS_CNT`.
6. После окончания строки выполняется добавление последнего токена (если он не пустой и удовлетворяет порогу длины).

3.5 Обработка UTF-8 и нормализация

Для поддержки кириллицы используются следующие решения:

- `utf8_len` определяет длину символа по старшему байту UTF-8;
- `is_valid_utf8` проверяет, что продолжения символа имеют вид `10xxxxxx`;
- `is_cyrillic` ограничивает допустимые 2-байтовые последовательности диапазоном русских букв;
- `utf8_to_lower` приводит кириллицу к нижнему регистру, а также нормализует “Ё/ё” к “е” (это упрощает дальнейшую обработку и совпадает с частой практикой индексирования).

Ограничение текущей реализации: токенизатор ориентирован на русский текст и поддерживает только ASCII-латиницу и кириллицу (2 байта). Другие письменности и символы выступают разделителями.

3.6 Использование в построении индекса

Функция `tokenize` применяется при построении словаря терминов и postings-листов (см. модуль `builder.cpp`):

- из каждого документа извлекается текст;
- выполняется токенизация;
- далее каждый токен передаётся в модуль стемминга, после чего учитывается в статистиках (частоты, df).

4 Стемминг

4.1 Теоретические сведения

Стемминг (stemming) — это приведение слова к его *основе* (stem) путём удаления типичных словоизменительных и словообразовательных суффиксов. Цель стемминга в информационном поиске — повысить полноту (recall): разные формы одного слова (“картина”, “картины”, “картиной”) должны попадать в один и тот же термин индекса.

Стемминг отличается от лемматизации:

- стемминг — эвристический, быстрый, может давать “не-словарную” основу;
- лемматизация — пытается восстановить словарную форму (лемму), но обычно требует морфологического словаря и более тяжёлой обработки.

4.2 Реализация и структура программы

Стемминг реализован в файлах:

- `stemming.hpp/.cpp` — функция `stem_word` и набор правил;
- `utf8_utils.hpp/.cpp` — вычисление длины UTF-8 символа, используется для обхода строки по символам.

Основной интерфейс:

```
std::string stem_word(std::string token);
```

Важное правило реализации: если токен начинается с ASCII-байта (например, англ. слова, числа), он возвращается без изменений. Стемминг применяется только к русским словам в UTF-8.

4.3 Алгоритм стемминга (русский Porter/Snowball-подход)

Используемый алгоритм соответствует классической схеме Porter/Snowball для русского языка и основан на понятии *регионов* слова:

- **RV** — часть слова после первой гласной;
- **R1** — часть слова после первого перехода “гласная–согласная” (VC);
- **R2** — часть слова после первого перехода VC внутри R1.

Регионы нужны, чтобы не удалять суффиксы “слишком рано” и не портить основу слова.

В реализации вычисление регионов выполняется функцией `mark_regions`, которая проходит слово по UTF-8 символам и использует список русских гласных.

Далее применяются последовательные шаги (в соответствии с кодом `stemming.cpp`):

4.3.1 Шаг 1. Удаление словоизменительных суффиксов в RV

На первом шаге удаляются наиболее вероятные окончания в следующем порядке:

1. **Perfective gerund** (деепричастные окончания) — сначала по правилам с условием “предшествует `а` или `я`”, затем без условия.
2. **Reflexive** — удаление возвратных окончаний `ся`/`сь`.
3. **Adjectival** — удаление прилагательных окончаний; затем, если прилагательное было удалено, дополнительно удаляются окончания причастий.
4. **Verb** — если прилагательное не подошло, пробуется удалить глагольные окончания (возможны варианты с условием по `а`/`я`).
5. **Noun** — если глагольные тоже не подошли, пробуется удалить именные окончания.

4.3.2 Шаг 2. Удаление конечной `и` в RV

Если слово оканчивается на `и` и эта буква находится в регионе RV, она удаляется.

4.3.3 Шаг 3. Derivational суффиксы в R2

Суффиксы словообразования (`ость`/`ост`) удаляются только в регионе R2, чтобы снизить риск переусечения.

4.3.4 Шаг 4. Финальная нормализация

На завершающем шаге выполняются:

- удаление суперлативов (`ейше`/`ейш`);
- устранение удвоенной `нн` (сводится к одной `н`) в RV;
- удаление мягкого знака `ь` в RV (если другие правила не сработали).

4.4 Особенности UTF-8 в реализации

Русские буквы в UTF-8 кодируются двумя байтами, и реализация учитывает это:

- обход строки выполняется шагом `utf8_len`;
- проверки вида “предыдущая буква = а/я” реализованы через сравнение 2-байтовых подпоследовательностей;
- алгоритм предполагает, что токены уже приведены к нижнему регистру и нормализованы (в т.ч. “ё” может быть приведена к “е” на этапе токенизации).

4.5 Использование в построении индекса

Стемминг применяется сразу после токенизации (см. `builder.cpp`):

- токен → стем `stem_word(token)`;
- **частота термина** (tf) накапливается в `terms_cnt[stem]++`;
- **частота документов** (df) считается через добавление стема в множество на документ (`unordered_set stems`), чтобы один документ учитывался в postings-листе термина только один раз.

5 Закон Ципфа и проверка на корпусе

5.1 Теоретические сведения

Закон Ципфа описывает распределение частот слов в естественных языках: если отсортировать термины по убыванию частоты встречаемости и присвоить им ранг $r = 1, 2, 3, \ldots$, то частота $f(r)$ приближённо подчиняется степенному закону:

$$f(r) \approx \frac{C}{r^s}, \tag{1}$$

где $C > 0$ — нормировочная константа, а показатель s для реальных текстов обычно близок к 1. В логарифмических координатах зависимость (1) становится линейной:

$$\log f(r) = \log C - s \log r, \tag{2}$$

то есть на графике $\log f$ от $\log r$ точки должны располагаться около прямой с углом наклона $-s$.

Данное свойство часто используется как быстрый sanity-check качества корпуса и предобработки: после токенизации и стемминга в корректно собранном корпусе распределение частот обычно демонстрирует близость к закону Ципфа. Теоретические сведения приведены по [2, 1].

5.2 Подготовка данных (таблица частот)

Для построения графика использовалась заранее подготовленная таблица частот (CSV-файл `zipf.csv`), сформированная на этапе построения индекса:

1. из каждого очищенного документа извлекался текст;
2. выполнялась токенизация и нормализация регистра;
3. применялся стемминг (приведение словоформ к основе);
4. накапливалась частота термина $freq$ (общее число вхождений по корпусу);
5. термины сортировались по убыванию $freq$, после чего каждому присваивался ранг $rank$ (начиная с 1).

Таким образом, входными данными для анализа является таблица вида:

```
rank, freq
```

где $rank$ — порядковый номер термина в отсортированном списке, $freq$ — его частота.

5.3 Построение графика

Построение выполнялось скриптом `plot_zipf.py`. Скрипт:

- читает `zipf.csv` из каталога экспорта `EXPORT_DIR` (по умолчанию `/exports`);
- проверяет наличие столбцов `rank` и `freq`;
- строит график в логарифмических координатах: `plt.loglog(rank, freq)`.

Для наглядности на график добавлены две аппроксимации:

1. **Теоретическая кривая Ципфа** с $s = 1$ и $C = f(1)$:

$$f_{\text{theory}}(r) = f(1) \cdot r^{-1}. \tag{3}$$

2. **Оценка показателя s по данным корпуса** методом линейной регрессии в логарифмическом пространстве. Для выбранного диапазона рангов $r \in [10,\ 10000]$ выполняется аппроксимация (2):

$$y = a + bx, \quad x = \log r,\ y = \log f(r), \tag{4}$$

после чего $s = -b$, $C = e^a$.

Использование диапазона рангов (с отсечением первых самых частых слов и хвоста распределения) позволяет уменьшить влияние:

- наиболее частых служебных слов и специфики тематики корпуса (“голова” распределения);
- редких слов и единичных вхождений (hapax legomena), чувствительных к размеру корпуса (“хвост”).

5.4 Результаты и интерпретация

Полученный график представлен на рисунке 1. По результатам аппроксимации показатель степени составил:

$$s \approx 1.038,$$

что близко к классическому значению $s \approx 1$ и подтверждает ожидаемое Zipf-поведение распределения частот в корпусе.

Отклонения точек от прямой в области малых рангов (самые частые термины) и больших рангов (редкие термины) являются типичными и объясняются:

- ограниченным объёмом корпуса;
- особенностями предобработки (токенизация, нормализация, стемминг);
- тематикой корпуса (высокая частота некоторых терминов, связанных с искусством).

5.5 Иллюстрация

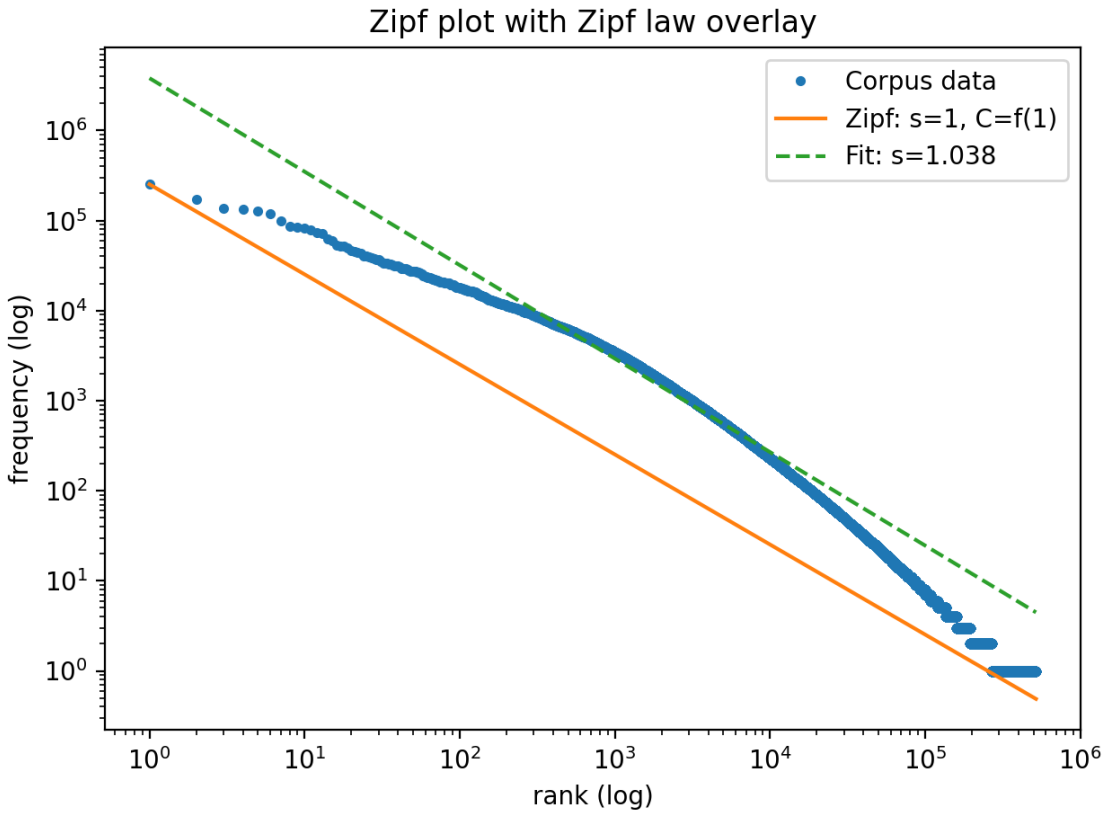

Рис. 1: График Ципфа для корпуса: зависимость частоты термина $freq$ от ранга $rank$ в логарифмических координатах. Точки — данные корпуса; сплошная линия — теоретический закон Ципфа при $s = 1$ и $C = f(1)$; пунктирная линия — аппроксимация по данным корпуса ($s \approx 1.038$).

6 Булевый индекс

6.1 Теоретические сведения

Булевый (логический) поиск — классическая модель информационного поиска, в которой запрос задаётся логическим выражением над терминами, а результатом является *множество документов*, которые удовлетворяют этому выражению. Базовая интерпретация строится на **матрице инцидентности** "термин–документ":

$$M_{t,d} = \begin{cases} 1, & \text{если термин } t \text{ встречается в документе } d, \\ 0, & \text{иначе.} \end{cases}$$

Тогда для каждого термина t можно определить множество документов

$$P(t) = \{\, d \mid M_{t,d} = 1 \,\},$$

которое называется **списком вхождений** (postings list).

Операции булевого поиска задаются через операции над множествами:

$$P(A \texttt{ AND } B) = P(A) \cap P(B), \quad P(A \texttt{ OR } B) = P(A) \cup P(B), \quad P(\texttt{NOT } A) = U \setminus P(A),$$

где U — универсум документов коллекции.

Хранить матрицу M напрямую неэффективно (она разреженная), поэтому на практике используется **инвертированный индекс**:

$$t \;\to\; P(t),$$

то есть отображение из термина в список идентификаторов документов, где он встречается [2, 1].

6.2 Построение булевого индекса в работе

Построение индекса выполняется программой `builder.cpp`. Источник документов — коллекция MongoDB с очищенными текстами (`MONGO_COLLECTION_CLEAN`), где для каждого документа доступно поле `clean_text`.

6.2.1 Предобработка текста

Для каждого документа выполняется стандартный пайплайн предобработки:

1. извлечение текста `clean_text`;
2. токенизация (разделение на слова с поддержкой UTF-8 кириллицы);

3. нормализация регистра (приведение к нижнему регистру);

4. стемминг (приведение словоформ к основе).

На выходе получаются нормализованные термины (стемы), которые используются как ключи индекса.

6.2.2 TF и DF

В процессе построения индекса вычисляются две статистики:

- **частота термина** (TF) — общее число вхождений термина по корпусу:

$$tf(t) = \sum_{d \in U} \#(t, d),$$

  где $\#(t, d)$ — число вхождений термина t в документе d;

- **частота документов** (DF) — число документов, содержащих термин:

$$df(t) = |P(t)|.$$

В реализации:

- TF накапливается счётчиком `terms_cnt[stem]++` для каждого вхождения;

- DF реализуется через множество `unordered_set stems` на документ: если стем впервые встретился в текущем документе, идентификатор документа добавляется в postings-лист (`terms_dict[stem].push_back(doc_id)`).

Таким образом, в postings-листе отсутствуют повторы одного и того же документа.

6.2.3 Упорядоченность postings-листов

Булевы операции объединения/пересечения/разности далее реализованы через "слияние" двух отсортированных списков двумя указателями. Для корректной и быстрой работы важно, чтобы списки документов были **отсортированы**. В работе это обеспечивается следующим образом:

- документы в MongoDB перебираются в порядке возрастания `_id`;

- каждый postings-лист пополняется в порядке чтения документов, следовательно, IDs внутри postings-листа идут в возрастающем порядке.

6.3 Форматы выходных файлов индекса

Индекс и вспомогательные данные сохраняются в каталог `EXPORT_DIR` (задаётся переменной окружения).

6.3.1 Файл postings (`POSTINGS_FILE`)

Postings-листы сохраняются в CSV-файл со структурой:

```
term,df,docs
```

где:

- `term` — стем (ключ индекса);
- `df` — количество документов, содержащих термин;
- `docs` — список идентификаторов документов, объединённый в строку через разделитель `|`.

6.3.2 Список всех документов (`DOCS_LIST`)

Для реализации операции `NOT` требуется универсум документов U. Он сохраняется отдельным файлом `DOCS_LIST` в виде одной строки:

```
id_1|id_2|...|id_N
```

где каждый `id` — строковое представление `ObjectId` длиной 24 символа.

6.3.3 Таблица частот для анализа (Zipf)

Дополнительно сохраняется файл `ZIPF_FILE` со структурой:

```
rank,term,freq
```

где `freq` соответствует $tf(t)$, а `rank` — рангу термина при сортировке по убыванию частоты.

6.4 Сложность операций над postings-листами

Для отсортированных списков A и B булевы операции выполняются за линейное время:

$$A \cap B,\ A \cup B,\ A \setminus B \ \Rightarrow\ O(|A| + |B|).$$

Это достигается “слиянием” двумя индексами (аналогично merge-шагу в сортировке слиянием): на каждом шаге сравниваются текущие элементы списков и один из указателей сдвигается.

7 Булевый поиск

7.1 Теоретические сведения

Булевый поиск обрабатывает запрос как логическое выражение над терминами (литералами). Поддерживаются операторы:

- `AND` — конъюнкция (пересечение множеств документов);
- `OR` — дизъюнкция (объединение множеств документов);
- `NOT` — отрицание (дополнение множества относительно U);
- круглые скобки `( )` для задания порядка вычислений.

В классической модели приоритеты операций задаются так:

`NOT` $>$ `AND` $>$ `OR`.

В реализованном парсере это зафиксировано явно: `NOT` имеет максимальный приоритет и является правоассоциативным оператором, `AND` имеет средний приоритет, `OR` — минимальный.

7.2 Ограничения и валидатор запроса

Терминальный интерфейс намеренно использует строгий разбор (“строгую токенизацию” запроса), чтобы избежать неоднозначностей и мусорных символов.

Разрешены:

- слова, состоящие из латинских букв/цифр (ASCII) или кириллицы (UTF-8);
- пробелы (как разделители);
- круглые скобки `(` и `)`.

Запрещены любые другие символы (например, дефис “-”, кавычки, запятые и т.п.). При наличии запрещённого символа выводится сообщение:

```
[ERROR] Запрещённый символ в запросе на позиции i.
```

Также запрещён символ `!` внутри запроса. Для выхода предусмотрена команда `!q` (строго отдельной строкой). Если в строке встречается `!`, но строка не равна `!q`, выводится подсказка:

```
[ERROR] '!' запрещён. Для выхода введи !q.
```

7.3 Построение обратной польской записи (RPN)

Для вычисления булевого выражения запрос сначала преобразуется из инфиксной формы (обычной записи) в **обратную польскую запись** (RPN). Преобразование выполняется алгоритмом "сортировочной станции" (Shunting-yard):

1. термины отправляются сразу в выходную очередь;
2. операторы помещаются в стек с учётом приоритета и ассоциативности;
3. скобки управляют извлечением операторов из стека.

Дополнительно реализована проверка синтаксиса (состояния "ожидается операнд / ожидается оператор"):

- запрещены пустые скобки `()`;
- выражение не может заканчиваться оператором;
- `NOT` должен стоять перед термином или `(`.

7.4 Вычисление запроса по RPN

После получения RPN запрос вычисляется стековым алгоритмом:

1. Если токен — термин, то:
   - выполняется токенизация и стемминг термина;
   - из файла postings извлекается postings-лист $P(t)$ и кладётся в стек.
2. Если токен — `AND` / `OR`, то из стека снимаются два множества документов и выполняется пересечение/объединение.
3. Если токен — `NOT`, то из стека снимается одно множество документов и выполняется дополнение относительно U (используется заранее загруженный файл `DOCS_LIST`).

В реализации используются операции над **отсортированными векторами** идентификаторов документов:

- `b_intersect(a,b)` — пересечение $A \cap B$;
- `b_union(a,b)` — объединение $A \cup B$;
- `b_diff(a,b)` — разность $A \setminus B$ (в запросе применяется как $U \setminus A$ для `NOT`).

7.5 Вывод результатов (просмотр документов)

После вычисления итогового множества документов:

- печатается общее число найденных документов;
- выводятся первые 5 результатов (или меньше, если ответ короткий);
- для каждого результата по `_id` запрашиваются метаданные из MongoDB: `title`, `url_norm`, `url_clean`, `source_name`;
- формируется сниппет — первые 100 UTF-8 символов поля `clean_text` с добавлением "..." при усечении.

7.6 Пример использования

Ниже приведён пример диалога с терминальным интерфейсом поисковика.

```
Добро пожаловать в поисковик 'как и почему это работает'!
терминал принимает только слова из кириллицы и латиницы!! Остальные знаки запрещены!!
Булевые операнды (только в таком регистре): AND OR NOT ) (
Для Выхода используйте !q или ^C
Да пребудет с Вами сила!
дизайн

-Результаты поиска-
Найдено всего документов: 1482
Первые 5:

[1] 697160fb88e85843dc7a2c21
title: Ёж Соник
url:   https://ru.wikipedia.org/wiki/%D0%81%D0%B6_%D0%A1%D0%BE%D0%BD%D0%B8%D0%BA
url_clean:   https://ru.wikipedia.org/wiki/Ёж_Соник
src:   ruwiki
snip:  У этого термина существуют и другие значения,см. Соник . Эта статья
о персонаже; другие значения: S...

[2] 697160fb88e85843dc7a2c39
title: Ёсида,Акихико
url:   https://ru.wikipedia.org/wiki/%D0%81%D1%81%D0%B8%D0%B4%D0%B0%2C_%D0%90%D0%BA%D0
url_clean:   https://ru.wikipedia.org/wiki/Ёсида,_Акихико
src:   ruwiki
```

```
snip:  В Википедии есть статьи о других людях с фамилией Ёсида . Акихико Ёсида
Дата рождения 15 февра...

[3] 697160fb88e85843dc7a2c4d
title: Аалто,Алвар
url:   https://ru.wikipedia.org/wiki/%D0%90%D0%B0%D0%BB%D1%82%D0%BE%2C_%D0%90%D0%BB%D0
url_clean:   https://ru.wikipedia.org/wiki/Аалто,_Алвар
src:   ruwiki
snip:  В Википедии есть статьи о других людях с фамилией Аалто . Алвар Аалто
фин. Alvar Aalto Основные свед...

[4] 697160fd88e85843dc7a2c9f
title: Абель де Пюжоль,Александр Дени
url:   https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B5_%D0%9
url_clean:   https://ru.wikipedia.org/wiki/Абель_де_Пюжоль,_Александр_Дени
src:   ruwiki
snip:  В Википедии есть статьи о других людях с фамилией Абель . Абель де Пюжоль
фр. Abel de Pujol Фотограф...

[5] 697160fd88e85843dc7a2ccf
title: Абстрактный иллюзионизм
url:   https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%
url_clean:   https://ru.wikipedia.org/wiki/Абстрактный_иллюзионизм
src:   ruwiki
snip:  Абстрактный иллюзионизм - направление искусства,сочетающее абстракцию
и иллюзию . Впервые термин уп...

-дизайн AND искусство
[ERROR] Запрещённый символ в запросе на позиции 2.
дизайн AND искусство

-Результаты поиска-
Найдено всего документов: 856
Первые 5:

[1] 697160fb88e85843dc7a2c39
title: Ёсида,Акихико
url:   https://ru.wikipedia.org/wiki/%D0%81%D1%81%D0%B8%D0%B4%D0%B0%2C_%D0%90%D0%BA%D0
url_clean:   https://ru.wikipedia.org/wiki/Ёсида,_Акихико
src:   ruwiki
snip:  В Википедии есть статьи о других людях с фамилией Ёсида . Акихико Ёсида
```

```
Дата рождения 15 февра...

[2] 697160fb88e85843dc7a2c4d
title: Аалто,Алвар
url:   https://ru.wikipedia.org/wiki/%D0%90%D0%B0%D0%BB%D1%82%D0%BE%2C_%D0%90%D0%BB%D0
url_clean:   https://ru.wikipedia.org/wiki/Аалто,_Алвар
src:   ruwiki
snip:  В Википедии есть статьи о других людях с фамилией Аалто . Алвар Аалто
фин. Alvar Aalto Основные свед...

[3] 697160fd88e85843dc7a2c9f
title: Абель де Пюжоль,Александр Дени
url:   https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B5_%D0%9
url_clean:   https://ru.wikipedia.org/wiki/Абель_де_Пюжоль,_Александр_Дени
src:   ruwiki
snip:  В Википедии есть статьи о других людях с фамилией Абель . Абель де Пюжоль
фр. Abel de Pujol Фотограф...

[4] 697160fd88e85843dc7a2ccf
title: Абстрактный иллюзионизм
url:   https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%
url_clean:   https://ru.wikipedia.org/wiki/Абстрактный_иллюзионизм
src:   ruwiki
snip:  Абстрактный иллюзионизм - направление искусства,сочетающее абстракцию
и иллюзию . Впервые термин уп...

[5] 697160ff88e85843dc7a2d3c
title: Аватар: Легенда об Аанге
url:   https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D0%B0%D1%82%D0%B0%D1%80:_%D0%9B%D0%
url_clean:   https://ru.wikipedia.org/wiki/Аватар:_Легенда_об_Аанге
src:   ruwiki
snip:  У этого термина существуют и другие значения,см. Аватар: Легенда об
Аанге (значения) . Термин «Ават...

-дизайн AND NOT музей

-Результаты поиска-
Найдено всего документов: 1039
Первые 5:

[1] 697160fb88e85843dc7a2c21
```

```
title: Ёж Соник
url:   https://ru.wikipedia.org/wiki/%D0%81%D0%B6_%D0%A1%D0%BE%D0%BD%D0%B8%D0%BA
url_clean:   https://ru.wikipedia.org/wiki/Ёж_Соник
src:   ruwiki
snip:  У этого термина существуют и другие значения,см. Соник . Эта статья
о персонаже; другие значения: S...

[2] 697160fb88e85843dc7a2c39
title: Ёсида,Акихико
url:   https://ru.wikipedia.org/wiki/%D0%81%D1%81%D0%B8%D0%B4%D0%B0%2C_%D0%90%D0%BA%D0
url_clean:   https://ru.wikipedia.org/wiki/Ёсида,_Акихико
src:   ruwiki
snip:  В Википедии есть статьи о других людях с фамилией Ёсида . Акихико Ёсида
Дата рождения 15 февра...

[3] 697160fd88e85843dc7a2cd9
title: Абсурдопедия
url:   https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%83%D1%80%D0%B4%D0%BE%D0%BF%
url_clean:   https://ru.wikipedia.org/wiki/Абсурдопедия
src:   ruwiki
snip:  Абсурдопедия URL См. « Ссылки » Коммерческий нет Тип сайта вики -энциклопедия
Регистрация необязате...

[4] 697160fe88e85843dc7a2d09
title: Авалон (фильм,1990)
url:   https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D0%B0%D0%BB%D0%BE%D0%BD_(%D1%84%D0%B
url_clean:   https://ru.wikipedia.org/wiki/Авалон_(фильм,_1990)
src:   ruwiki
snip:  У этого термина существуют и другие значения,см. Авалон (значения) .
Авалон англ. Avalon Жанр драма...

[5] 697160ff88e85843dc7a2d3c
title: Аватар: Легенда об Аанге
url:   https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D0%B0%D1%82%D0%B0%D1%80:_%D0%9B%D0%B
url_clean:   https://ru.wikipedia.org/wiki/Аватар:_Легенда_об_Аанге
src:   ruwiki
snip:  У этого термина существуют и другие значения,см. Аватар: Легенда об
Аанге (значения) . Термин «Ават...

-(дизайн OR кино) AND NOT мультфильм
[ERROR] Запрещённый символ в запросе на позиции 5.
```

```
[ERROR] Пустой запрос.
(дизайн OR кино) AND NOT мультфильм

-Результаты поиска-
Найдено всего документов: 4682
Первые 5:

[1] 697160fa88e85843dc7a2c07
title: «Квартет И» по Амстелу
url:   https://ru.wikipedia.org/wiki/%C2%AB%D0%9A%D0%B2%D0%B0%D1%80%D1%82%D0%B5%D1%82
url_clean:   https://ru.wikipedia.org/wiki/«Квартет_И»_по_Амстелу
src:   ruwiki
snip:  «Квартет И» по Амстелу Жанр комедия Режиссёр Ярослав Чеважевский [вд]
Сценаристы Леонид Барац Ростис...

[2] 697160fa88e85843dc7a2c11
title: «Спартак». Действующие лица и... болельщики
url:   https://ru.wikipedia.org/wiki/%C2%AB%D0%A1%D0%BF%D0%B0%D1%80%D1%82%D0%B0%D0%BA%
url_clean:   https://ru.wikipedia.org/wiki/«Спартак»._Действующие_лица_и..._болельщик
src:   ruwiki
snip:  «Спартак»: действующие лица... и болельщики Жанры документальный ,спорт
Режиссёры Илья Гутман ,Иосиф...

[3] 697160fa88e85843dc7a2c15
title: ×××HOLiC
url:   https://ru.wikipedia.org/wiki/%C3%97%C3%97%C3%97HOLiC
url_clean:   https://ru.wikipedia.org/wiki/×××HOLiC
src:   ruwiki
snip:  ×××HOLiC Обложка первого тома манги яп. ××× ×××Horikku ( ромадзи ) ×××Хорикку
( киридзи ) Жанр /...

[4] 697160fb88e85843dc7a2c25
title: Ёлки-палки!..
url:   https://ru.wikipedia.org/wiki/%D0%81%D0%BB%D0%BA%D0%B8-%D0%BF%D0%B0%D0%BB%D0%BA
url_clean:   https://ru.wikipedia.org/wiki/Ёлки-палки!..
src:   ruwiki
snip:  О сети ресторанов см. Ёлки-палки . Ёлки-палки!.. Жанр комедия Режиссёр
Сергей Никоненко Автор сценар...

[5] 697160fb88e85843dc7a2c27
```

```
title: Ёлки последние
url:   https://ru.wikipedia.org/wiki/%D0%81%D0%BB%D0%BA%D0%B8_%D0%BF%D0%BE%D1%81%D0%BB
url_clean:   https://ru.wikipedia.org/wiki/Ёлки_последние
src:   ruwiki
snip:  Ёлки последние Жанр комедия Режиссёры Тимур Бекмамбетов Егор Баранов
Анна Пармас Александр Котт Нико...

!q
```

8 Выводы

Выполнив лабораторную работу по курсу «Информационный поиск», я последовательно построил полный минимальный пайплайн получения корпуса, предобработки текста и выполнения булевого поиска по документам. По итогам работы можно сделать следующие выводы.

- Я разобрался, как организовать **воспроизводимый процесс** подготовки данных и вычислений: конфигурация вынесена в `config.yaml` и `.env`, а запуск этапов оформлен через Docker Compose и Makefile. Это позволяет повторять эксперимент на другой машине без ручной настройки окружения.
- Я освоил получение исходного списка документов (seeds) из **официальных дампов Wikimedia** через MySQL: скачивание `.sql.gz`, импорт в БД, проверка таблиц и контроль повторного импорта через маркеры. Практически это показывает, как строится стабильная основа для дальнейшей добычи корпуса без зависимости от случайной выдачи сайта.
- Я реализовал формирование тематического набора документов по категории «Искусство»: обход графа категорий (BFS) до заданной глубины, фильтрация редиректов и нецелевых namespace, ограничение размера списка и балансировка по источникам. В результате получен контролируемый набор URL для дальнейшего сбора.
- Я реализовал **поискового робота** для загрузки HTML по seeds и сохранения данных в MongoDB с метаданными и механизмами устойчивости (таймауты, повторные попытки, корректная обработка ограничений, возобновление по checkpoint). Это даёт практический навык построения краулинга, который не разваливается при первом же сетевом сбое.
- Я добавил этап **очистки HTML** (`html_cleaner`): выделение основного контента страницы, удаление служебных блоков и получение чистого текста для индексирования. Важный вывод: качество поиска почти всегда упирается в качество предобработки, а не в “магический алгоритм”.
- Я реализовал **токенизацию** с поддержкой UTF-8 кириллицы и нормализацией регистра, а также продумал правила, какие символы считаются частью токена, а какие разделителями. Это научило аккуратно работать с кодировками и не ломать текст “по байтам”.
- Я реализовал **стемминг** русских слов (правила Porter/Snowball-подхода) и увидел, как нормализация словоформ повышает полноту поиска: разные формы одного слова начинают сопоставляться с одним термином индекса.

- Я построил **булевый индекс** (инвертированный индекс) в виде отображения `term` → `postings list` и сформировал файлы экспорта (postings, список всех документов для операции `NOT`, таблица частот). Я понял практическую разницу между TF (общее число вхождений) и DF (число документов с термином), и зачем обе статистики полезны.

- Я реализовал **булевый поиск**: строгую валидацию запроса, преобразование выражения в RPN (алгоритм сортировочной станции), вычисление результата через операции над отсортированными postings-листами (пересечение/объединение/разность). Вывод: булев поиск прост по идее, но требует аккуратной инженерии в парсинге и обработке ошибок ввода.

- Я проверил корпус на соответствие **закону Ципфа**: построил лог-лог график "ранг–частота" по подготовленной таблице частот и получил показатель степени близкий к 1 (в моём случае $s \approx 1.038$). Это подтвердило адекватность корпуса и предобработки (токенизация/стемминг) на уровне статистического поведения текста.

- Я понял ограничения булевого поиска: он не ранжирует результаты по релевантности и чувствителен к точному набору терминов. Следующий естественный шаг развития системы — добавить ранжирование (TF-IDF/BM25), стоп-слова, позиционный индекс для фразового поиска и более "мягкую" обработку запросов пользователя.

Полученные навыки напрямую применимы при разработке поисковых систем и корпоративного поиска, в задачах анализа текстовых коллекций, построения индексов для документации/вики, а также как базовый фундамент для более продвинутых моделей ранжирования и извлечения информации.

Список литературы

[1] Manning, C. D., Raghavan, P., Schütze, H. *Introduction to Information Retrieval*. — Cambridge University Press, 2008.

[2] Маннинг, К. Д., Рагхаван, П., Шютце, Х. *Введение в информационный поиск*. — М.: Издательский дом «Вильямс», 2011.

[3] Jurafsky, D., Martin, J. H. *Speech and Language Processing*. — 2nd ed. (draft). — URL: https://web.stanford.edu/~jurafsky/slp3/ (дата обращения: 6 февраля 2026 г.).

[4] The Unicode Consortium. *The Unicode Standard*. — URL: https://www.unicode.org/standard/standard.html (дата обращения: 6 февраля 2026 г.).

[5] Yergeau, F. *RFC 3629: UTF-8, a transformation format of ISO 10646*. — IETF, 2003. — URL: https://www.rfc-editor.org/rfc/rfc3629 (дата обращения: 6 февраля 2026 г.).

[6] Porter, M. F. An algorithm for suffix stripping. *Program*, 1980, Vol. 14(3), pp. 130–137.

[7] Snowball (Porter) stemming algorithms. Russian stemming algorithm. — URL: https://snowballstem.org/algorithms/russian/stemmer.html (дата обращения: 6 февраля 2026 г.).

[8] ГОСТ Р 7.0.5–2008. Библиографическая ссылка. Общие требования и правила составления.