МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Лабораторная работа
по курсу «Информационный поиск»

Выполнил: Васильев Д. А.
Группа: М8О-403Б-22
Преподаватель: Кухтичев А.А.

Москва, 2025

ЦЕЛЬ РАБОТЫ

● Найти исторические документы из русской Википедии для работы

● Подготовить документы к дальнейшему использованию: выделить текст и заголовок, убрать разметку и ненужную информацию

● Привести несколько примеров запросов к существующим поисковикам, указать недостатки в полученной поисковой выдаче

● Реализовать простой обратный индекс с использованием кастомных структур данных

● Реализовать сохранение индекса в памяти для быстрого поиска

● Реализовать поиск по набору документов с использованием булевой логики (&&, ||, !)

ОПИСАНИЕ ДАННЫХ

В качестве источника данных был выбран набор статей из русской Википедии по исторической тематике. Этот ресурс хранит тысячи статей и имеет открытое API, что позволяет легко получить доступ к нужным нам документам. Однако, выгрузка всей Википедии может занять достаточно много времени, поэтому было принято решение взять лишь категории, связанные с историей.

Для получения данных был написан скрипт на Python, который обращается к открытому API рекурсивно и забирает нужную информацию, а затем записывает каждую статью в MongoDB коллекцию.

```
def fetch_category_members(cat, cmcontinue=None):
    params = {
        "action": "query",
        "list": "categorymembers",
        "cmtitle": cat,
        "cmtype": "page|subcat",
        "cmlimit": "max",
        "format": "json",
    }
    if cmcontinue:
        params["cmcontinue"] = cmcontinue

    r = session.get(API_URL, params=params, timeout=30)
    r.raise_for_status()
    data = r.json()
```

```
    members = data["query"]["categorymembers"]
    next_token = data.get("continue", {}).get("cmcontinue")
    return members, next_token
```

Листинг 1 - функция получения содержимого категории

Функция `fetch_category_members` является основной для обхода категорий (см. рис. 1), она по имени категории забирает список страниц и подкатегорий, затем рекурсивно обрабатывает их и сохраняет статьи в базу данных.

После парсинга Википедии мы имеем коллекцию в MongoDB, где содержатся наши документы, которые имеют следующие характеристики:

- Количество документов - 49 810
- Общий объём текста - 640 577 045 байт
- Средний размер текста - 12 863 символа
- Общее количество токенов - 44 563 861

Пример структуры документа:

```
{
  "url": "https://ru.wikipedia.org/?curid=12345",
  "text": "Вторая мировая война — крупнейший вооружённый
конфликт...",
  "title": "Вторая мировая война",
  "pageid": 12345,
  "created_at": 12345678,
  "source": "ru.wikipedia.org"
}
```

Листинг 2 - пример структуры документа

График распределения частот терминов (Закон Ципфа):

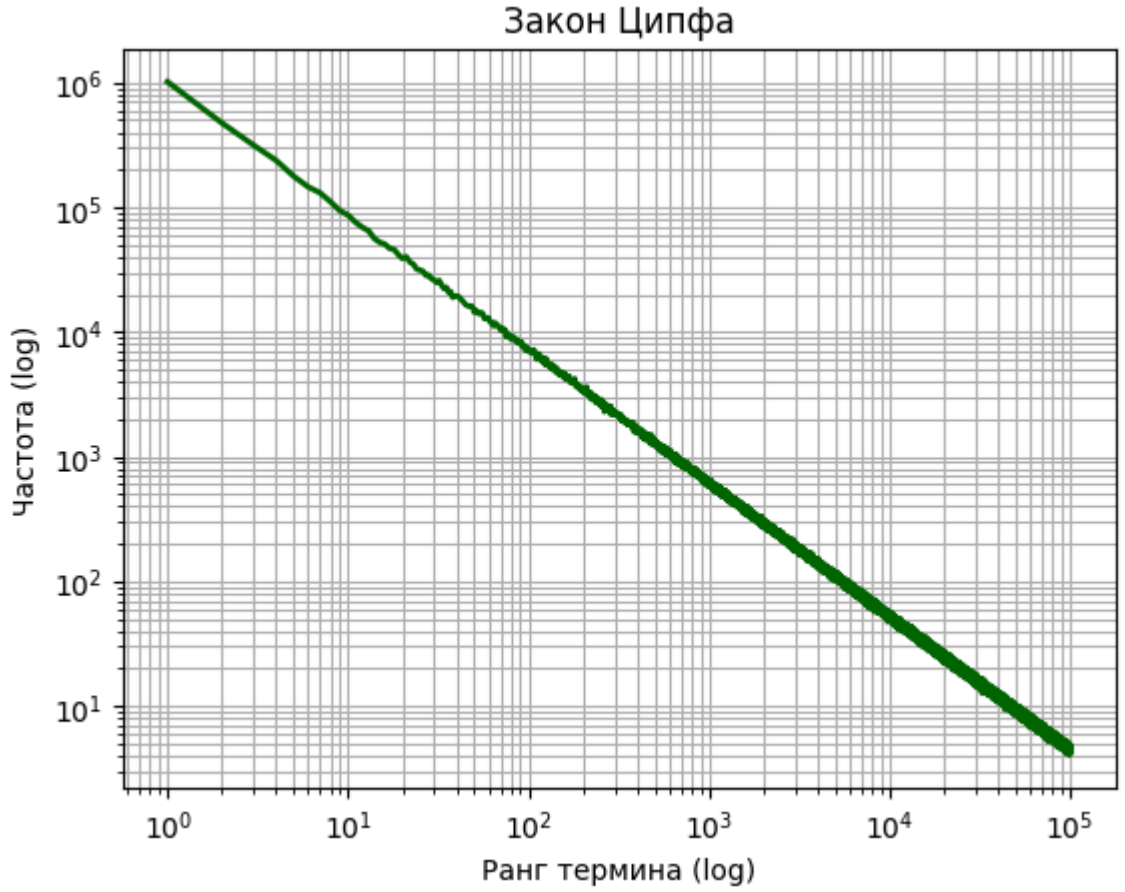

Рисунок 1- Закон Ципфа

ПРИМЕРЫ СУЩЕСТВУЮЩИХ ПОИСКОВИКОВ

Самыми главными недостатками существующих поисковиков являются:

● Игнорирование потребностей пользователей: Поисковые системы могут требовать от пользователей высокого уровня знания и навыков для составления правильного запроса, особенно при поиске исторической информации.

● Отражение только популярных источников: Алгоритмы поисковых систем часто ориентируются на популярность источников, игнорируя менее известные, но более релевантные для конкретного исторического запроса.

● Сложное управление информацией: Увеличение объёма доступной информации делает сложнее организовать и структурировать результаты поиска по историческим темам.

● Отсутствие специализации: Общие поисковые системы не учитывают специфику исторической терминологии и хронологических связей между событиями.

ИНДЕКСАЦИЯ И ПОИСК

Поисковый индекс — структура данных, которая содержит информацию о документах и используется в поисковых системах. Обратный индекс хранит отображение слов на документы, в которых они встречаются.

В этой работе мы делаем индексирование следующим способом:

● Выбираем документ из MongoDB
● Извлекаем заголовок и текст
● Для каждого слова в тексте производим токенизацию и стемминг
● В обратный индекс для слова добавляем идентификатор документа

```
std::wstring StemRu(const std::wstring& word) {
    if (word.length() <= kMinWordLength)
        return word;

    for (const auto& end : kEndings) {
        if (word.length() > end.length() + kMinStemLength &&
            word.compare(word.length() - end.length(), end.length(),
end) == 0) {
            return word.substr(0, word.length() - end.length());
        }
    }
    return word;
}
```

Листинг 3 - функция стемминга русского языка

```
containers::HashSet<DocID> BooleanSearchRu(const std::string& query,
InvertedIndex& index) {
    auto tokens = text_processing::TokenizeQuery(query);

    if (tokens.empty()) {
        return containers::HashSet<DocID>();
    }

    QueryParser parser(tokens);
    return parser.Parse(index);
}
```

Листинг 4 - функция булева поиска

Описание поиска:

● Пользователь отправляет POST-запрос с JSON телом, содержащим поле "query"
● Сервер парсит запрос для обработки операций (&&, ||,!)
● Далее находим документы, которые содержат в себе слова запроса с учётом операций
● После того как получили набор документов, возвращаем их пользователю

Веб-сервис предоставляет три эндпоинта:
- `GET /health` - проверка работоспособности
- `GET /stats` - статистика индекса

- `POST /search` - выполнение поискового запроса

```
void Server::Start() {
    auto* server = new httplib::Server();
    server_impl_ = server;

    server->Get("/health", [this](const httplib::Request&,
httplib::Response& res) {
        res.set_content(HandleHealth(), kContentTypeText);
    });

    server->Get("/stats", [this](const httplib::Request&,
httplib::Response& res) {
        res.set_content(HandleStats(), kContentTypeJson);
    });

    server->Post("/search", [this](const httplib::Request& req,
httplib::Response& res) {
        auto query_opt = ParseJsonQuery(req.body);
        if (!query_opt.has_value()) {
            res.status = 400;
            res.set_content(CreateErrorResponse("Invalid JSON or
missing 'query' field"), kContentTypeJson);
            return;
        }
        res.set_content(HandleSearch(query_opt.value()),
kContentTypeJson);
    });

    std::cout << "Server starting on port " << port_ << std::endl;
    server->listen("0.0.0.0", port_);
}
```

Листинг 5 - запуск веб-сервера

ПРИМЕР РАБОТЫ

Запрос 1:

```
curl -X POST localhost:8080/search -d '{"query": "Петр || Екатерина"}'
```


```
{
      "count" : 4308,
      "documents" :
      [
            {
                  "created_at" : 1766612671,
                  "id" : "694b056ef52f4014029c9c70",
                  "pageid" : 128334,
                  "title" : "Исторический анекдот",
                  "url" : "https://ru.wikipedia.org/?curid=128334"
            },
            {
                  "created_at" : 1766612671,
                  "id" : "694b056ef52f4014029c9c86",
                  "pageid" : 1769409,
                  "title" : "Униформология",
                  "url" : "https://ru.wikipedia.org/?curid=1769409"
            },
            {
                  "created_at" : 1766612671,
                  "id" : "694b056ef52f4014029c9c88",
                  "pageid" : 227387,
                  "title" : "Фаворитизм",
                  "url" : "https://ru.wikipedia.org/?curid=227387"
            },
       ...
    ]
}
```


Листинг 6 - пример работы запроса к сервису поиска

Запрос 2:

```
curl -X POST localhost:8080/search -d '{"query": "Война && Мир"}'
{
      "count" : 290,
      "documents" :
      [
            {
                  "created_at" : 1766612672,
                  "id" : "694b056ef52f4014029c9d86",
                  "pageid" : 1714363,
                  "title" : "Вторая англо-афганская война",
                  "url" : "https://ru.wikipedia.org/?curid=1714363"
            },
            {
                  "created_at" : 1766612672,
                  "id" : "694b056ef52f4014029c9da6",
                  "pageid" : 6117433,
                  "title" : "Архитектура Парижа в эпоху абсолютизма",
                  "url" : "https://ru.wikipedia.org/?curid=6117433"
            },
            {
                  "created_at" : 1766612672,
                  "id" : "694b056ff52f4014029c9e84",
                  "pageid" : 9136512,
                  "title" : "История Мичигана",
                  "url" : "https://ru.wikipedia.org/?curid=9136512"
            },
            ...
       ]
}
```

Листинг 7 - пример работы запроса к сервису поиска

Статистика поисковика:

- Время составления индекса: 278 секунд
- Среднее время обработки запроса: 45 миллисекунд
- Количество индексированных документов: 49 810
- Общий размер индекса в памяти: ~1.2 ГБ

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы реализации алгоритмов поиска информации и индексирования были изучены основные принципы работы поисковых систем, алгоритмы обработки естественного языка, а также процесс индексации. Разработана система, которая эффективно индексирует исторические тексты на русском языке и предоставляет возможность выполнения сложных булевых запросов через REST API. Система демонстрирует высокую производительность при обработке запросов и может быть использована в образовательных и исследовательских целях для работы с историческими документами.