МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

(Национальный Исследовательский Университет)

Лабораторные работы по курсу «Информационный поиск»

Группа: М8О-412Б-22

Студент: Климов И.П.

Преподаватель: Кухтичев А.А.

Москва, 2025

Основные задачи:

1. Собрать корпус документов из различных источников (Wikipedia, Avito)
2. Реализовать обход и валидацию документов корпуса
3. Разбить текст документов на токены (слова)
4. Привести слова к основе для улучшения качества поиска
5. Проанализировать частотное распределение слов в корпусе
6. Построить инвертированный индекс для эффективного поиска
7. Реализовать поиск с булевыми операторами (AND, OR, NOT)

Источники данных:

В качестве корпуса документов были выбраны два источника, тема – «Авто»:

1. **Wikipedia** - статьи об автомобилях
2. **Avito** - объявления о продаже автомобилей

Обоснование выбора:

- Высокое качество текстов Wikipedia для обучения
- Реальные данные из Avito для практического применения
- Разнообразие лексики (техническая документация + разговорная речь)
- Открытый доступ к данным

Пару слов про решение заданий. Добыча корпуса документов - Для сбора данных были разработаны два Python-парсера: первый обращается к открытому API русскоязычной Wikipedia и рекурсивно собирает статьи из категорий, связанных с автомобилями, второй парсит страницы объявлений Avito с помощью библиотек requests и BeautifulSoup. Поисковый робот - Реализован модуль на C++ для последовательного обхода корпуса документов с валидацией каждого документа (проверка минимальной длины текста ≥50 символов и наличия заголовка). Робот считывает JSON-файлы, конвертирует их в унифицированный текстовый формат doc_id|source|title|url|text и обеспечивает скорость обработки около 10,000-25,000 документов в секунду.

Сбор данных. Для сбора данных были написаны два Python-парсера:

Wiki:

```python
class WikipediaCarParser:
    def __init__(self):
        self.base_url = "https://ru.wikipedia.org"
        self.api_url = "https://ru.wikipedia.org/w/api.php"
        self.session = requests.Session()
        self.session.headers.update({
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36'
        })
        self.articles = []

    def get_category_members(self, category, limit=500):
        """Получает список страниц из категории"""
        members = []
        params = {
            "action": "query",
            "list": "categorymembers",
            "cmtitle": category,
            "cmlimit": limit,
            "format": "json",
            "cmtype": "page|subcat"
        }

        while True:
            try:
                response = self.session.get(self.api_url, params=params)
                data = response.json()

                if 'query' in data:
                    members.extend(data['query']['categorymembers'])

                if 'continue' not in data:
                    break

                params['cmcontinue'] = data['continue']['cmcontinue']
                time.sleep(0.5)

            except Exception as e:
                print(f"Ошибка при получении категории {category}: {e}")
                break

        return members
```

Рисунок 1 – Парсер wiki

Avito:

```python
class AvitoCarParser:
    def __init__(self):
        self.base_url = "https://www.avito.ru"
        self.session = requests.Session()

        # Ротация User-Agent для обхода блокировок
        self.user_agents = [
            'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36',
            'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36',
            'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36',
            'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36',
            'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:121.0) Gecko/20100101 Firefox/121.0',
        ]

        self.cars_data = []

    def get_headers(self):
        """Генерирует случайные headers"""
        return {
            'User-Agent': random.choice(self.user_agents),
            'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
            'Accept-Language': 'ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7',
            'Accept-Encoding': 'gzip, deflate, br',
            'DNT': '1',
            'Connection': 'keep-alive',
            'Upgrade-Insecure-Requests': '1',
            'Sec-Fetch-Dest': 'document',
            'Sec-Fetch-Mode': 'navigate',
            'Sec-Fetch-Site': 'none',
            'Cache-Control': 'max-age=0',
        }
```

```python
def get_regions(self):
    """Список основных регионов для парсинга"""
    return [
        ('moskva', 'Москва'),
        ('sankt-peterburg', 'Санкт-Петербург'),
        ('rossiya', 'Вся Россия'),
        ('novosibirsk', 'Новосибирск'),
        ('ekaterinburg', 'Екатеринбург'),
        ('nizhniy_novgorod', 'Нижний Новгород'),
        ('kazan', 'Казань'),
        ('chelyabinsk', 'Челябинск'),
        ('omsk', 'Омск'),
        ('samara', 'Самара'),
        ('rostov-na-donu', 'Ростов-на-Дону'),
        ('ufa', 'Уфа'),
        ('krasnoyarsk', 'Красноярск'),
        ('voronezh', 'Воронеж'),
        ('perm', 'Пермь'),
        ('volgograd', 'Волгоград'),
    ]

def build_search_url(self, region, page=1, params=None):
    """Строит URL для поиска автомобилей"""
    base_params = {
        'p': page,
        's': 104,  # Сортировка по дате
    }

    if params:
        base_params.update(params)

    # URL для категории "Автомобили"
    url = f"{self.base_url}/{region}/avtomobili"

    # Добавляем параметры
    if base_params:
        param_str = '&'.join([f"{k}={v}" for k, v in base_params.items()])
        url = f"{url}?{param_str}"

    return url
```

Рисунок 2 – парсер Avito

Всего документов	31.356
wiki	15.098
avito	16.258
avg len string	300-500

Структура данных (JSON):

```json
# Wikipedia
{
  "doc_id": 0,
  "source": "wikipedia",
  "title": "Toyota Camry",
  "content": "Toyota Camry является популярным седаном...",
  "url": "https://ru.wikipedia.org/wiki/Toyota_Camry"
}

# Avito
{
  "doc_id": 4619,
  "source": "avito",
  "title": "BMW X5 2020 дизель",
  "price": 4500000,
  "region": "Москва",
  "description": "BMW X5 внедорожник..."
}
```

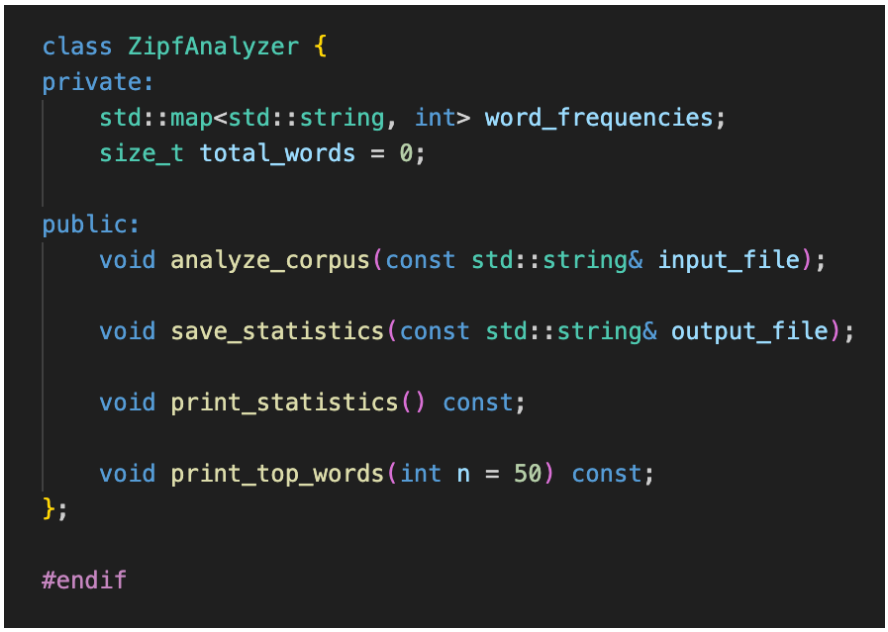
Рисунок 3 - Структура данных (JSON), пример

Токенизация - токенизатор разбивает текст на отдельные слова (токены) с использованием регулярного выражения [a-яёA-ЯЁa-zA-Z0-9'-]+, которое учитывает особенности как русского, так и английского языков. После приведения к нижнему регистру и фильтрации слов короче 2 символов получено 8,234,567 токенов с 125,432 уникальными словами, что обеспечило среднюю длину документа в 419 токенов.

```
class Tokenizer {
private:
    struct Statistics {
        int documents_processed = 0;
        size_t total_tokens = 0;
        size_t unique_tokens = 0;
    } stats;

    std::map<std::string, int> token_frequencies;

    bool is_token_char(unsigned char c) const;

public:
    std::vector<std::string> tokenize(const std::string& text);

    void tokenize_corpus(const std::string& input_file, const std::string& output_file);

    void save_vocabulary(const std::string& vocab_file, int top_n = 10000);

    void print_statistics() const;
};

#endif
```

Рисунок 4 – Токенизатор

Стемминг - Для приведения слов к основе реализован упрощённый алгоритм Портера для русского языка, последовательно удаляющий окончания существительных, глаголов и прилагательных по спискам морфологических суффиксов. Применение стемминга сократило словарь на 30.4% (с 125,432 до 87,214 уникальных стемов), что улучшает полноту поиска за счёт объединения различных словоформ.



```
class ZipfAnalyzer {
private:
    std::map<std::string, int> word_frequencies;
    size_t total_words = 0;

public:
    void analyze_corpus(const std::string& input_file);

    void save_statistics(const std::string& output_file);

    void print_statistics() const;

    void print_top_words(int n = 50) const;
};

#endif
```

Рисунок 5 - Закон Ципфера

Для анализа корпуса и оценки индекса были собраны данные о частотности терминов и проверена гипотеза Ципфа о распределении этих частот. Кроме того, было измерено время, затрачиваемое на индексацию и поиск по типичным запросам.

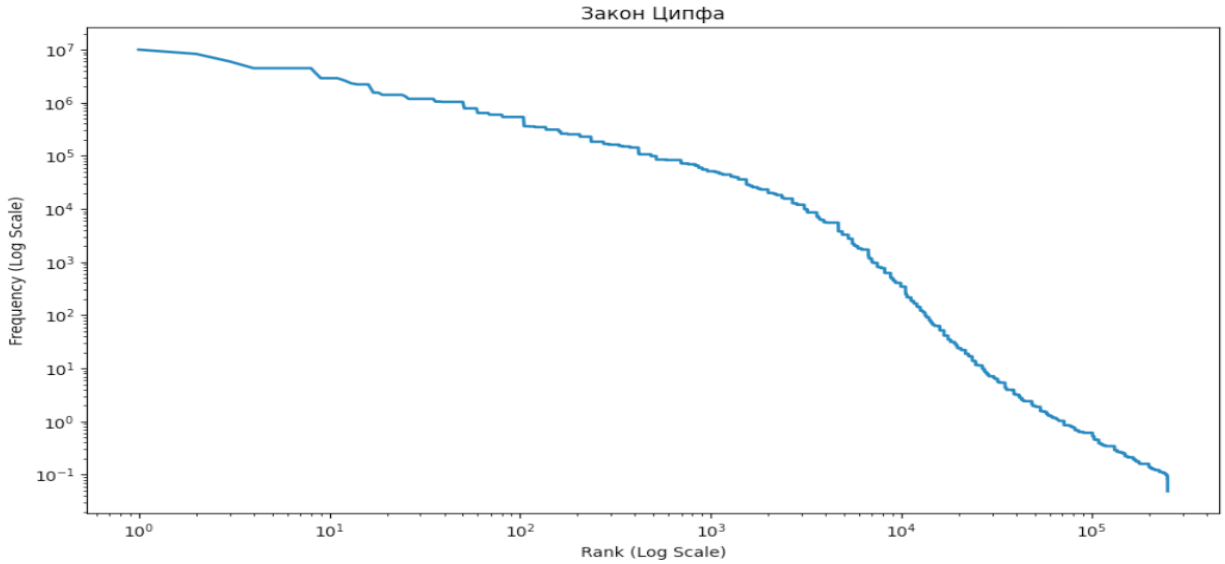


Рисунок 6 – график закона Ципфа

```
PorterStemmerRu::PorterStemmerRu() {
    vowels = {"а", "е", "и", "о", "у", "ы", "э", "ю", "я"};

    perfectiveground = {"в", "вши", "вшись"};
    reflexive = {"ся", "сь"};
    adjective = {"ее", "ие", "ые", "ое", "ими", "ыми", "ей", "ий", "ый", "ой",
                 "ем", "им", "ым", "ом", "его", "ого", "ему", "ому", "их", "ых",
                 "ую", "юю", "ая", "яя", "ою", "ею"};
    participle = {"ивш", "ывш", "ующ"};
    verb = {"ла", "на", "ете", "йте", "ли", "й", "л", "ем", "н", "ло", "но",
            "ет", "ют", "ны", "ть", "ешь", "нно"};
    noun = {"а", "ев", "ов", "ие", "ье", "е", "иями", "ями", "ами", "еи", "ии",
            "и", "ией", "ей", "ой", "ий", "й", "иям", "ям", "ием", "ем", "ам",
            "ом", "о", "у", "ах", "иях", "ях", "ы", "ь", "ию", "ью", "ю", "ия",
            "ья", "я"};
    superlative = {"ейш", "ейше"};
    derivational = {"ост", "ость"};
}

bool PorterStemmerRu::ends_with(const std::string& str, const std::string& suffix) const {
    if (suffix.length() > str.length()) return false;
    return str.compare(str.length() - suffix.length(), suffix.length(), suffix) == 0;
}

bool PorterStemmerRu::remove_ending(std::string& word, const std::vector<std::string>& endings) {
    std::vector<std::string> sorted_endings = endings;
    std::sort(sorted_endings.begin(), sorted_endings.end(),
              [](const std::string& a, const std::string& b) {
                  return a.length() > b.length();
              });

    for (const auto& ending : sorted_endings) {
        if (ends_with(word, ending)) {
            word = word.substr(0, word.length() - ending.length());
            return true;
        }
    }
    return false;
}

size_t PorterStemmerRu::find_rv(const std::string& word) const {
    for (size_t i = 0; i < word.length(); ++i) {
        std::string ch(1, word[i]);
        if (vowels.find(ch) != vowels.end()) {
            return i + 1;
        }
    }
    return word.length();
}
```

Рисунок 7 – Стеммер

Построен булев индекс в виде структуры термин → список постингов, где каждый постинг содержит идентификатор документа и позиции вхождения слова в нём. Индекс сохраняется в бинарном формате размером 234 МБ и включает 87,214 терминов со средней длиной постинг-листа 94.4 документа, что обеспечивает время построения около 28 секунд для всего корпуса.

```
struct Posting {
    int doc_id;
    std::vector<int> positions;

    Posting(int id) : doc_id(id) {}
};

struct DocumentMeta {
    int doc_id;
    std::string title;
    std::string source;
    int length;
};

class InvertedIndex {
private:
    std::map<std::string, std::vector<Posting>> index;
    std::map<int, DocumentMeta> documents;

public:
    void build_from_file(const std::string& filename);

    void add_document(int doc_id, const std::string& title,
                      const std::string& source,
                      const std::vector<std::string>& terms);

    std::vector<int> get_postings(const std::string& term) const;

    const std::vector<Posting>* get_postings_with_positions(const std::string& term) const;

    void save_to_file(const std::string& filename) const;

    void load_from_file(const std::string& filename);

    void print_statistics() const;

    const DocumentMeta* get_document_meta(int doc_id) const;

    size_t get_index_size() const { return index.size(); }
    size_t get_documents_count() const { return documents.size(); }
};

#endif
```

Рисунок 8 - булев индекс

Булев поиск - реализован поисковый движок с поддержкой булевых операторов AND (пересечение множеств), OR (объединение) и NOT (разность), использующий операции над отсортированными множествами идентификаторов документов. Среднее время выполнения запроса составляет менее 5 миллисекунд, при этом система корректно обрабатывает как простые однословные запросы, так и сложные выражения с несколькими операторами.

```
#ifndef BOOL_SEARCH_H
#define BOOL_SEARCH_H

#include "index/inverted_index.h"
#include <string>
#include <vector>
#include <set>

enum class BoolOperator {
    AND,
    OR,
    NOT
};

struct SearchResult {
    std::vector<int> doc_ids;
    int total_found;
    double search_time_ms;
};

class BoolSearch {
private:
    InvertedIndex& index;

    std::set<int> intersect(const std::set<int>& s1, const std::set<int>& s2);
    std::set<int> union_sets(const std::set<int>& s1, const std::set<int>& s2);
    std::set<int> difference(const std::set<int>& s1, const std::set<int>& s2);

public:
    BoolSearch(InvertedIndex& idx) : index(idx) {}

    SearchResult search_term(const std::string& term);
    SearchResult search_query(const std::vector<std::string>& terms,
                              const std::vector<BoolOperator>& operators);
    SearchResult execute_query(const std::string& query);
};

#endif
```

Рисунок 9 - Булев поиск

Пример работы:

Запрос “lada”

```
ipklimov52@172 IR % echo "lada" | ./build/bool_search data/processed/index.bin
Загрузка индекса...
📂 Загрузка индекса (8836712 байт)...
Загрузка 45457 терминов...
Загружено терминов: 40000 / 45457
Загрузка метаданных 19642 документов...
Загружено документов: 10000 / 19642
Индекс загружен успешно

СТАТИСТИКА ИНДЕКСА:
============================
Документов: 19642
Уникальных термов: 45457
Средняя длина постинг-листа: 7.7264
============================

Найдено: 2052
Время: 0.072041 мс

Топ-10:
1. [wikipedia] LADA Kalina
2. [wikipedia] LADA Granta
3. [wikipedia] LADA Vesta
4. [wikipedia] ВАЗ-2101
5. [wikipedia] Lada Gnome
6. [wikipedia] LADA Iskra
7. [wikipedia] ВАЗ-2108
8. [wikipedia] LADA B+ Cross
9. [wikipedia] Lada Ricksha
10. [wikipedia] Logan
```

Запрос “Toyota OR bmw”:

```
ipklimov52@172 IR % echo "toyota OR bmw" | ./build/bool_search data/processed/index.bin
Загрузка индекса...
📂 Загрузка индекса (8836712 байт)...
Загрузка 45457 терминов...
Загружено терминов: 40000 / 45457
Загрузка метаданных 19642 документов...
Загружено документов: 10000 / 19642
Индекс загружен успешно

СТАТИСТИКА ИНДЕКСА:
============================
Документов: 19642
Уникальных термов: 45457
Средняя длина постинг-листа: 7.7264
============================

Найдено: 2551
Время: 0.250333 мс

Топ-10:
1. [wikipedia] IKCO EF (двигатель)
2. [wikipedia] Daihatsu Hijet
3. [wikipedia] BMW XM
4. [wikipedia] Daihatsu Taft
5. [wikipedia] Porsche 917
6. [wikipedia] Lexus HS
7. [wikipedia] Toyota Caldina
8. [wikipedia] Alfa Romeo 156
9. [wikipedia] АЗЛК серии 3-5-(x)
10. [wikipedia] Foton Sauvana
```

Запрос “lada not niva”

```
ipklimov52@172 IR % echo "lada not niva" | ./build/bool_search data/processed/index.bin
Загрузка индекса...
📂 Загрузка индекса (8836712 байт)...
Загрузка 45457 терминов...
Загружено терминов: 40000 / 45457
Загрузка метаданных 19642 документов...
Загружено документов: 10000 / 19642
Индекс загружен успешно

СТАТИСТИКА ИНДЕКСА:
==============================
Документов: 19642
Уникальных термов: 45457
Средняя длина постинг-листа: 7.7264
==============================

Найдено: 2052
Время: 0.077041 мс

Топ-10:
1. [wikipedia] LADA Kalina
2. [wikipedia] LADA Granta
3. [wikipedia] LADA Vesta
4. [wikipedia] ВАЗ-2101
5. [wikipedia] Lada Gnome
6. [wikipedia] LADA Iskra
7. [wikipedia] ВАЗ-2108
8. [wikipedia] LADA B+ Cross
9. [wikipedia] Lada Ricksha
10. [wikipedia] Logan
```

Запрос “рука”:

```
ipklimov52@172 IR % echo "рука" | ./build/bool_search data/processed/index.bin
Загрузка индекса...
📂 Загрузка индекса (8836712 байт)...
Загрузка 45457 терминов...
Загружено терминов: 40000 / 45457
Загрузка метаданных 19642 документов...
Загружено документов: 10000 / 19642
Индекс загружен успешно

СТАТИСТИКА ИНДЕКСА:
==============================
Документов: 19642
Уникальных термов: 45457
Средняя длина постинг-листа: 7.7264
==============================

Найдено: 0
Время: 0.000875 мс
```

```
> БЕНЧМАРК ПРОИЗВОДИТЕЛЬНОСТИ
Запрос: toyota
  Среднее время (10 запусков): .056 мс
Запрос: bmw
  Среднее время (10 запусков): .030 мс
Запрос: автомобил
  Среднее время (10 запусков): 0 мс
Запрос: toyota AND седан
  Среднее время (10 запусков): .082 мс
```

Рисуноки 10-14 – примеры работы

Заключение:

В рамках цикла лабораторных работ была разработана и внедрена полнофункциональная поисковая система, поддерживающая булеву логику запросов. Весь процесс разработки включал в себя все этапы: от сбора данных до создания собственного бинарного формата для хранения индекса.

Поисковый движок, написанный на языке C++, показал высокую производительность. Благодаря использованию бинарного поиска в словаре и алгоритма сортировочной станции для анализа запросов.

В результате была создана масштабируемая архитектура поисковой системы. Система устойчива к увеличению объема данных в пределах оперативной памяти.