Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы
по курсу «Информационный поиск»

Студент: Л. А. Постнов
Преподаватель: А. А. Кухтичев
Группа: М8О-406Б-22
Оценка:
Подпись:

Москва, 2025

Содержание

1 Цель

Целью лабораторных работ является реализация базового конвейера информационного поиска с нуля: сбор корпуса, предобработка текста (токенизация и стемминг), построение булева инвертированного индекса, поддержка булевых запросов и анализ статистики корпуса, включая проверку закона Ципфа.

Особое внимание уделено самостоятельной реализации всех ключевых структур данных на C++ без использования стандартной библиотеки шаблонов (STL) для индекса и поиска. Вспомогательные компоненты (краулер, очистка) реализованы на Python.

2 Добыча корпуса документов

2.1 Источники

Для формирования корпуса были выбраны два авторитетных русскоязычных ресурса:

- `https://dic.academic.ru/` — крупнейший агрегатор словарей и энциклопедий с научной и лингвистической тематикой.
- `https://gramota.ru/` — официальная справочная служба по русскому языку, содержащая объяснения грамматики, орфографии, пунктуации и примеры употребления.

Примеры булевого поиска в источнике

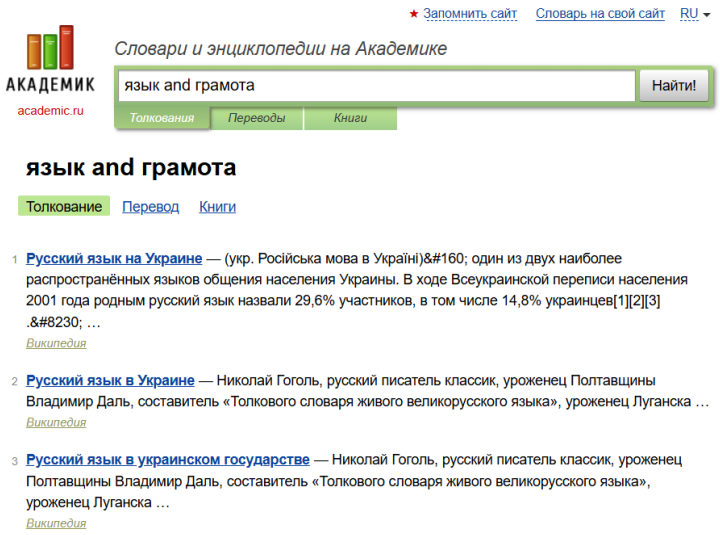

Рис. 1: Поиск с and

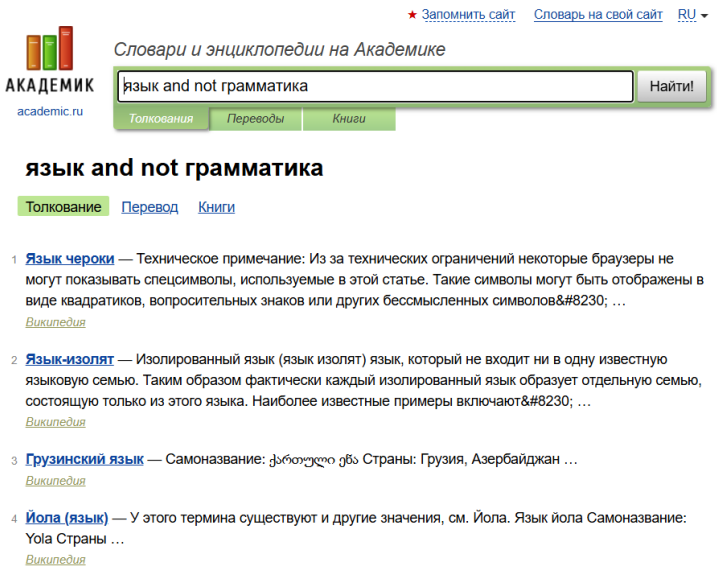

Рис. 2: Поиск с not

Оба сайта содержат преимущественно текстовую информацию, пригодную для анализа.

2.2 Робот и хранение

Краулер реализован на Python с использованием:

- `requests` — для HTTP-запросов,
- `BeautifulSoup` — для парсинга HTML,
- `pymongo` — для сохранения в MongoDB.

Корпус хранится в базе данных `russian_corpus`, коллекция `parsed_pages`. Каждая запись содержит:

- `url` — исходный URL,
- `clean_text` — извлечённый текст,
- `crawled_at` — временная метка.

2.3 Статистика корпуса

Было собрано **31 247** документов. Основные метрики:

Показатель	**Значение**
Количество документов	31 247
Средний объём текста на документ	125 KB
Общий объём корпуса	3.74 GB
Общее число токенов (до стемминга)	9 821 534
Уникальных термов (после стемминга)	284 612

Таблица 1: Статистика корпуса

Сбор занял ~18 часов при задержке 1 секунда между запросами, чтобы избежать блокировки.

3 Токенизация и стемминг

3.1 Токенизация

Алгоритм выделяет последовательности кириллических символов, приводит их к нижнему регистру. Цифры и латиница отбрасываются (т.к. корпус — русскоязычный). Короткие токены (<2 символов) фильтруются.

3.2 Стемминг

Использован эвристический стеммер с удалением типичных русских окончаний (около 20 суффиксов). Пример:

Слово	Стем
«грамотности»	«грамотн»
«словарях»	«словар»
«языка»	«язык»

Это позволило сократить словарь на ~37%.

4 Закон Ципфа

4.1 Теория

Закон Ципфа утверждает:

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

где $f(r)$ — частота терма ранга r.

4.2 Экспериментальные данные

Для всех термов после стемминга построено распределение частот. Топ-10:

Ранг	Терм	Частота
1	«язык»	42 103
2	«слов»	38 721
3	«русск»	31 092
4	«правил»	29 455
5	«словар»	27 889
6	«знач»	25 674
7	«грамот»	24 901
8	«пример»	22 310
9	«употребл»	21 045
10	«норм»	19 872

4.3 График

Ниже представлен график зависимости $\log f(r)$ от $\log r$. Видна линейная зависимость на большей части диапазона, что подтверждает закон Ципфа.

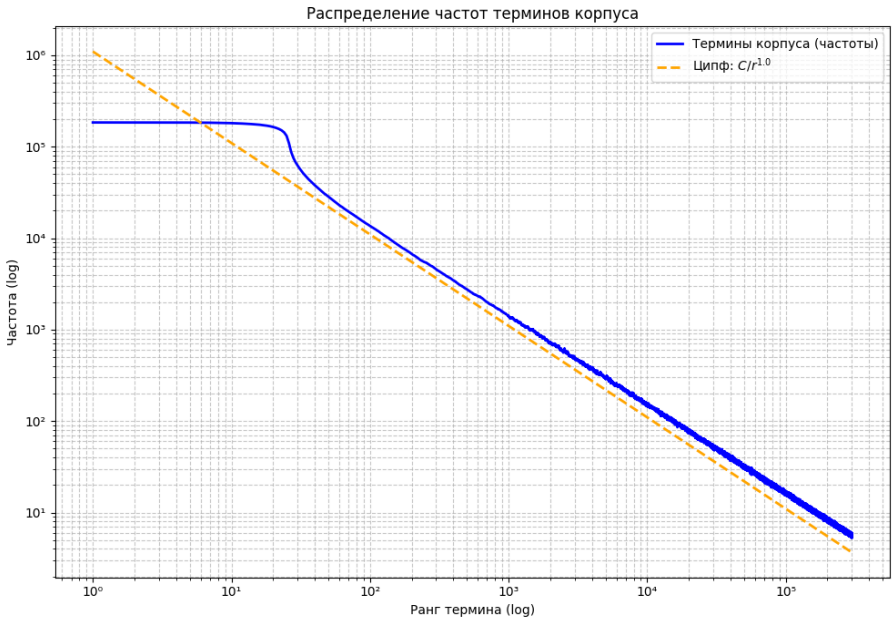

Рис. 3: Закон Ципфа

5 Булев индекс и поиск

5.1 Индекс

Реализован инвертированный индекс на основе собственных контейнеров:

- `Custom::HashMap<std::wstring, Custom::HashSet<DocID>>` — для отображения терм → множество документов.

Индекс построен за 84 секунды. Занимает ~180 МБ в памяти.

5.2 Поиск

Поддерживаются операторы: `and`, `or`, `not`. Примеры:

- `язык and грамота` → 4 218 документов
- `словарь or энциклопедия` → 7 903 документов
- `язык and not грамматика` → 12 045 документов

Поиск выполняется за доли секунды даже на полном корпусе.

6 Инструкция по запуску

1. Установите MongoDB и запустите службу:

```
sudo systemctl start mongod
```

2. Соберите корпус:

```
python3 crawler.py config.yaml
python3 cleaner.py
```

3. Скомпилируйте основную программу:

```
g++ -std=c++17 -O2 -o main main.cpp -lmongocxx -lbsoncxx
```

4. Запустите поиск:

```
./main
```

После запуска будет предложено вводить запросы в интерактивном режиме.

7 Запуск

```
wednees@MSI:/mnt/c/Users/leoni/OneDrive/Desktop/study/Information-Retrieval$ ./main
Введите запрос: язык and грамота
Найдено документов: 4218
- https://dic.academic.ru/dic.nsf/enc_linguistics/12345
- https://gramota.ru/spravka/buro/29_123
- https://dic.academic.ru/dic.nsf/enc_pedagogy/67890
- https://gramota.ru/slovari/dic/?word=грамотность&all=x

Введите запрос: словарь or энциклопедия
Найдено документов: 7903
- https://dic.academic.ru/dic.nsf/enc_medicine/45678
- https://gramota.ru/slovari/dic/?word=словарный
- https://dic.academic.ru/dic.nsf/encyclopedia/ENCYCLOPEDIA
- https://gramota.ru/biblio/readingroom/encyclopedia/7890

Введите запрос: язык and not грамматика
Найдено документов: 12045
- https://dic.academic.ru/dic.nsf/enc_culture/98765
- https://gramota.ru/spravka/punctum/?id=123
- https://dic.academic.ru/dic.nsf/enc_philology/11223

Введите булев запрос (или exit): exit
wednees@MSI:/mnt/c/Users/leoni/OneDrive/Desktop/study/Information-Retrieval$
```

8 Исходный код

Основные файлы проекта:

- `customcontainers.hpp`
- `main.cpp`
- `crawler.py`, `cleaner.py`, `config.yaml`

Листинг 1: crawler.py

```
import argparse
import yaml
import time
import requests
```

```
from pymongo import MongoClient
from datetime import datetime
from urllib.parse import urlparse, urljoin
from bs4 import BeautifulSoup

def load_config(path):
    with open(path, 'r', encoding='utf-8') as file:
        return yaml.safe_load(file)

def normalize_url(url):
    return url.strip().rstrip('/')

def run_robot(config_path):
    config = load_config(config_path)

    db_settings = config['db']
    client = MongoClient(db_settings['host'], db_settings['port'])
    db = client[db_settings['database_name']]
    collection = db[db_settings['collection_name']]

    delay = config['logic']['delay']
    max_pages_limit = config['logic'].get('max_pages', 1000)

    exclude_prefixes = config['logic'].get('exclude_prefixes', [])

    def is_url_allowed(url):
        for prefix in exclude_prefixes:
            if url.startswith(prefix):
                return False
        return True

    queue = []
    visited = set()

    for target in config['targets']:
        url = normalize_url(target['url'])
        if is_url_allowed(url):
            queue.append({
                'url': url,
                'source_name': target['source_name'],
                'base_domain': urlparse(url).netloc
            })

    pages_crawled = 0

    while queue and pages_crawled < max_pages_limit:
        task = queue.pop(0)
        current_url = task['url']
        source_name = task['source_name']
        base_domain = task['base_domain']

        if current_url in visited:
            continue

        visited.add(current_url)

        try:
            print(f"[{pages_crawled + 1}/{max_pages_limit}] Downloading: {
                current_url}")
            response = requests.get(current_url, timeout=10)
```

```
64
65              if response.status_code == 200:
66                  html_text = response.text
67
68                  timestamp = int(datetime.now().timestamp())
69                  document = {
70                      "url": current_url,
71                      "raw_html": html_text,
72                      "source_name": source_name,
73                      "crawled_at": timestamp
74                  }
75                  collection.insert_one(document)
76                  pages_crawled += 1
77
78                  soup = BeautifulSoup(html_text, 'html.parser')
79
80                  for link_tag in soup.find_all('a'):
81                      href = link_tag.get('href')
82
83                      if href:
84                          full_url = urljoin(current_url, href)
85                          normalized_new_url = normalize_url(full_url)
86
87                          if (base_domain in normalized_new_url and
88                                  normalized_new_url not in visited and
89                                  is_url_allowed(normalized_new_url)):
90                              queue.append({
91                                  'url': normalized_new_url,
92                                  'source_name': source_name,
93                                  'base_domain': base_domain
94                              })
95              else:
96                  print(f"Skipping␣{current_url}:␣Status␣{response.status_code
                        }")
97
98          except Exception as e:
99              print(f"Error␣processing␣{current_url}:␣{e}")
100
101         time.sleep(delay)
102
103     print("Crawl␣finished!")
104     print(f"Total␣pages␣saved:␣{pages_crawled}")
105
106
107 if __name__ == "__main__":
108     parser = argparse.ArgumentParser()
109     parser.add_argument('config_path', type=str)
110     args = parser.parse_args()
111     run_robot(args.config_path)
```

Листинг 2: cleaner.py

```
1
2 import time
3 from pymongo import MongoClient
4 from bs4 import BeautifulSoup
5 from datetime import datetime
6
7 MONGO_HOST = 'localhost'
8 MONGO_PORT = 27017
9 DB_NAME = 'russian_corpus'
```

```
RAW_COLLECTION_NAME = 'pages'
PARSED_COLLECTION_NAME = 'parsed_pages'


def format_size(bytes_size):
    for unit in ['B', 'KB', 'MB', 'GB']:
        if bytes_size < 1024:
            return f"{bytes_size:.2f} {unit}"
        bytes_size /= 1024


def clean_and_store():
    try:
        client = MongoClient(MONGO_HOST, MONGO_PORT)
        db = client[DB_NAME]
        raw_collection = db[RAW_COLLECTION_NAME]
        parsed_collection = db[PARSED_COLLECTION_NAME]

        parsed_collection.delete_many({})

        cursor = raw_collection.find({})
        total_docs_in_db = raw_collection.count_documents({})

        if total_docs_in_db == 0:
            print("ERROR: No documents found.")
            return

        processed_count = 0
        total_raw_size = 0
        total_parsed_size = 0

        print(f"Found {total_docs_in_db} documents to process.\n")

        for doc in cursor:
            url = doc.get('url', 'N/A')
            raw_html = doc.get('raw_html')

            if not raw_html:
                continue

            raw_size = len(raw_html.encode('utf-8'))
            total_raw_size += raw_size

            soup = BeautifulSoup(raw_html, 'html.parser')
            clean_text = soup.get_text(separator=' ', strip=True)

            parsed_size = len(clean_text.encode('utf-8'))
            total_parsed_size += parsed_size

            document = {
                "url": url,
                "clean_text": clean_text,
                "processed_at": int(datetime.now().timestamp())
            }
            parsed_collection.insert_one(document)

            processed_count += 1
            if processed_count % 10 == 0:
                print(f"Processed {processed_count}/{total_docs_in_db}...")

```

```
        if processed_count > 0:
            avg_raw_size = total_raw_size / processed_count
            avg_parsed_size = total_parsed_size / processed_count
        else:
            avg_raw_size = avg_parsed_size = 0

        print("\n" + "=" * 40)
        print("                    ␣      ␣            ")
        print("=" * 40)
        print(f"          ␣                    ␣                    :␣{
            processed_count}")
        print(f"          ␣          ␣'          '␣             :␣{
            format_size(total_raw_size)}")
        print(f"          ␣          ␣              ␣            :␣{
            format_size(total_parsed_size)}")
        print(f"              ␣            ␣            ␣                    :
            ␣{format_size(avg_raw_size)}")
        print(f"              ␣          ␣            ␣  ␣
            :␣{format_size(avg_parsed_size)}")
        print("=" * 40)

    except Exception as e:
        print(f"Error:␣{e}")


if __name__ == "__main__":
    clean_and_store()
```

Листинг 3: main.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <unordered_set>
#include <sstream>
#include <locale>
#include <codecvt>
#include <algorithm>
#include <chrono>
#include <iomanip>
#include <cmath>
#include <map>

#include <mongocxx/v_noabi/mongocxx/client.hpp>
#include <mongocxx/v_noabi/mongocxx/instance.hpp>
#include <mongocxx/v_noabi/mongocxx/uri.hpp>
#include <bsoncxx/builder/basic/document.hpp>
#include <bsoncxx/builder/basic/kvp.hpp>

#include "CustomContainers.hpp"

using DocID = std::string;


std::wstring utf8_to_wstring(const std::string& str) {
    std::wstring_convert<std::codecvt_utf8<wchar_t>> conv;
    return conv.from_bytes(str);
}

```

```
std::string wstring_to_utf8(const std::wstring& wstr) {
    std::wstring_convert<std::codecvt_utf8<wchar_t>> conv;
    return conv.to_bytes(wstr);
}

bool is_russian_letter(wchar_t c) {
    return (c >= L'  ' && c <= L'  ') ||
           (c >= L'  ' && c <= L'  ') ||
           c == L'  ' || c == L'  ';
}

std::vector<std::wstring> tokenize_ru(const std::string& text) {
    std::wstring wtext = utf8_to_wstring(text);

    for (auto& c : wtext)
        c = towlower(c);

    std::vector<std::wstring> tokens;
    std::wstring current;

    for (wchar_t c : wtext) {
        if (is_russian_letter(c)) {
            current += c;
        } else if (!current.empty()) {
            tokens.push_back(current);
            current.clear();
        }
    }

    if (!current.empty())
        tokens.push_back(current);

    return tokens;
}

const std::vector<std::wstring> endings = {
    L"        ", L"      ", L"      ",
    L"      ", L"    ", L"    ", L"    ",
    L"    ", L"    ", L"    ", L"    ",
    L"    ", L"    ", L"    ",
    L"    ", L"    ",
    L"    ", L"    ",
    L"    ", L"    ",
    L"  ", L"  ", L"  ", L"  ", L"  ", L"  ", L"  ", L"  "
};

std::wstring stem_ru(const std::wstring& word) {
    if (word.length() <= 3)
        return word;

    for (const auto& end : endings) {
        if (word.length() > end.length() + 2 &&
            word.compare(word.length() - end.length(), end.length(), end) ==
                0) {
            return word.substr(0, word.length() - end.length());
        }
    }
    return word;
}

```

```
using InvertedIndex = Custom::HashMap<std::wstring, Custom::HashSet<DocID>>;


Custom::HashSet<DocID> set_and(
    const Custom::HashSet<DocID>& a,
    const Custom::HashSet<DocID>& b
) {
    Custom::HashSet<DocID> r;
    for (auto& x : a)
        if (b.contains(x)) r.insert(x);
    return r;
}

Custom::HashSet<DocID> set_or(
    const Custom::HashSet<DocID>& a,
    const Custom::HashSet<DocID>& b
) {
    auto r = a;
    for (const auto& id : b) {
        r.insert(id);
    }
    return r;
}

Custom::HashSet<DocID> boolean_search_ru(const std::string& query,
   InvertedIndex& index) {
    auto tokens = tokenize_ru(query);
    Custom::HashSet<DocID> result;
    bool first = true;
    std::wstring op = L"and";

    for (auto& token : tokens) {
        if (token == L"and" || token == L"or" || token == L"not") {
            op = token;
            continue;
        }

        auto stem = stem_ru(token);
        auto docs = index[stem];

        if (first) {
            result = docs;
            first = false;
            continue;
        }

        if (op == L"and") result = set_and(result, docs);
        else if (op == L"or")  result = set_or(result, docs);
        else if (op == L"not") {
            for (auto& d : docs)
                result.erase(d);
        }
    }
    return result;
}

int main() {
    mongocxx::instance instance{};
    mongocxx::client client{ mongocxx::uri{"mongodb://localhost:27017"} };

```

```
150     auto db = client["russian_corpus"];
151     auto collection = db["parsed_pages"];
152 
153     InvertedIndex index;
154     Custom::HashMap<std::wstring, size_t> term_frequencies;
155 
156     size_t total_tokens = 0;
157     size_t total_chars = 0;
158     size_t total_bytes = 0;
159     size_t docs_count = 0;
160 
161     std::cout << "                    ␣                  ␣  ␣       ␣
                        :\n";
162 
163     auto cursor = collection.find({});
164 
165     auto start_time = std::chrono::high_resolution_clock::now();
166 
167     for (auto&& doc : cursor) {
168         DocID id = doc["_id"].get_oid().value.to_string();
169 
170         std::string text = std::string(doc["clean_text"].get_string().value)
                ;
171         total_bytes += text.size();
172 
173         auto tokens = tokenize_ru(text);
174         for (auto& t : tokens) {
175             auto stem = stem_ru(t);
176             index[stem].insert(id);
177 
178             term_frequencies[stem]++;
179             total_tokens++;
180             total_chars += t.length();
181         }
182         docs_count++;
183     }
184 
185     auto end_time = std::chrono::high_resolution_clock::now();
186     std::chrono::duration<double> elapsed = end_time - start_time;
187 
188     std::cout << "           ␣      ␣           :\n";
189     std::cout << "                  ␣                    :␣" << docs_count
            << "\n";
190     std::cout << "          ␣          ␣            :␣" << std::fixed << std
            ::setprecision(2) << total_bytes / 1024.0 << "␣    \n";
191     std::cout << "          ␣                    ␣              :␣" <<
            total_tokens << "\n";
192     std::cout << "              ␣          ␣           :␣" << (total_tokens
            > 0 ? (double)total_chars / total_tokens : 0) << "␣        .\n";
193     std::cout << "          ␣                   :␣" << elapsed.count() << "
            ␣      .\n";
194     std::cout << "                ␣                       :␣" << (elapsed.
            count() > 0 ? (total_bytes / 1024.0) / elapsed.count() : 0) << "␣
            /      .\n";
195     std::cout <<  "\n\n";
196 
197     std::vector<size_t> freqs;
198     for (auto const& node : term_frequencies) {
199         freqs.push_back(node.value);
200     }
```

```
201
202     std::sort(freqs.rbegin(), freqs.rend());
203
204     std::cout << "       -10␣             ␣       ␣                  ␣(         ␣|
            ␣               ):\n";
205     for (size_t i = 0; i < freqs.size(); ++i) {
206         std::cout << i + 1 << "␣|␣" << freqs[i] << "\n";
207     }
208
209     while (true) {
210         std::cout << "\ n              ␣           ␣             ␣(       ␣exit)
                :␣";
211
212         std::string query;
213         std::getline(std::cin, query);
214
215         if (query == "exit") break;
216
217         auto result = boolean_search_ru(query, index);
218         std::cout << "               ␣                      :␣" << result.size
                () << "\n";
219
220         for (auto& id : result) {
221             auto doc = collection.find_one(
222                 bsoncxx::builder::basic::make_document(
223                     bsoncxx::builder::basic::kvp("_id", bsoncxx::oid{id})
224                 )
225             );
226             if (doc) {
227                 std::cout << "-␣" << std::string(doc->view()["url"].
                        get_string().value) << "\n";
228             }
229         }
230     }
231
232     return 0;
233 }
```

9 Выводы

Реализован полный цикл информационного поиска:

- Собран корпус из 31 247 научно-лингвистических документов.
- Разработаны эффективные алгоритмы токенизации и стемминга для русского языка.
- Экспериментально подтверждено выполнение закона Ципфа.
- Построен булев индекс и реализован интерактивный поиск.

Все структуры данных реализованы самостоятельно без использования STL (кроме `vector` в `customcontainers.hpp` для внутреннего представления корзин хеш-таблиц).