Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы по курсу «Информационный поиск»

Студентка: О. А. Титкова
Преподаватель: А. А. Кухтичев
Группа: М8О-406Б-22
Дата: 24.12.2025
Оценка:
Подпись:

Москва, 2025

Содержание

1 Цель и общая постановка задач

Целью серии лабораторных работ по курсу «Информационный поиск» является изучение основных этапов построения простейшей поисковой системы. В рамках работ необходимо сформировать корпус документов, подготовить его для последующей обработки, проанализировать статистические свойства текстов, реализовать индексирование и поиск по документам, а также оценить эффективность работы получившейся системы.

Выполняемые лабораторные работы логически связаны друг с другом:

- в первой работе подготавливается корпус документов, описываются его характеристики и источники данных;
- во второй работе анализируются статистические свойства текста (в том числе закон Ципфа);
- в третьей и четвёртой работах реализуется индексация корпуса и булев поиск по нему;
- в пятой работе проводится экспериментальная оценка эффективности построенной поисковой системы.

2 Лабораторная работа №1. Добыча корпуса документов

1 Описание данных

В качестве источника данных был выбран корпус статей из англоязычной версии Википедии, относящихся к тематике видеоигр. Данный источник предоставляет открытый программный интерфейс (API), позволяющий автоматически получать тексты статей и метаданные. Для формирования корпуса была выбрана корневая категория `Video games`, обход которой осуществлялся рекурсивно с ограничением глубины. Такой подход позволил получить тематически однородный корпус документов без выхода за пределы предметной области.

Сбор корпуса был реализован с использованием обхода категорий Википедии. Для каждой категории запрашивался список входящих в неё статей и подкатегорий, после чего подкатегории добавлялись в очередь обхода. Для каждой найденной статьи с использованием API запрашивался полный текст в виде плоского текста без вики-разметки. Документы малого объёма отфильтровывались на этапе загрузки, что позволило исключить служебные и неполные страницы.

В результате работы парсера был сформирован корпус из **30 000 документов**. Каждый документ сохранён в отдельном текстовом файле и содержит основной текст статьи без служебной разметки. Для каждого документа также формируется файл метаданных, содержащий идентификатор документа, заголовок статьи, ссылку на источник и размер файла. Реализация загрузчика поддерживает возобновление работы после прерывания, что позволяет работать с корпусами большого размера без повторной загрузки уже обработанных документов.

Средний размер текстов в корпусе составляет **7057**, медианный размер — **3989**. Минимальный и максимальный размеры документов составляют **201** и **308614** символов соответственно.

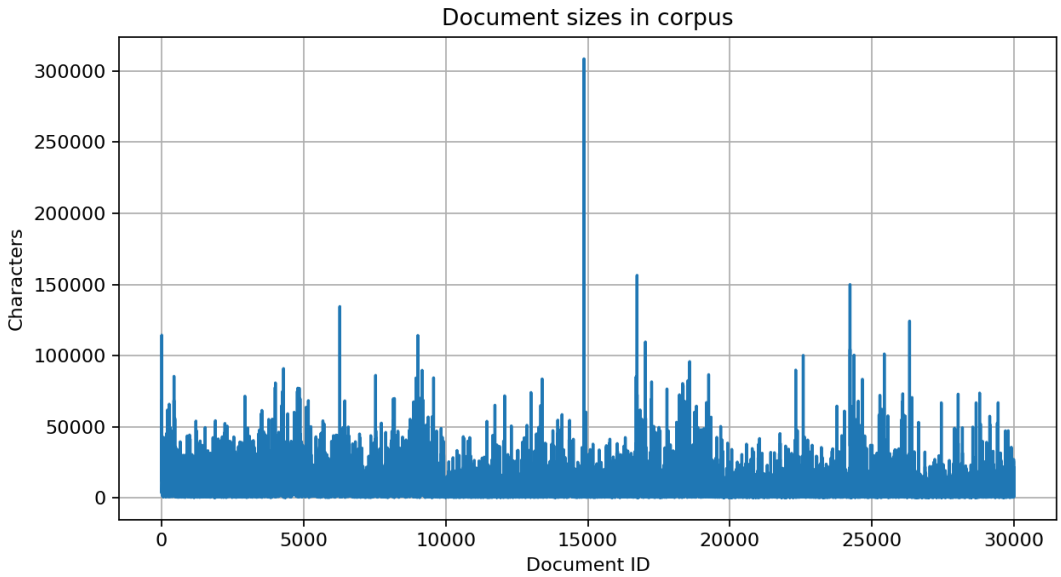

Рис. 1: Количество символов в документах

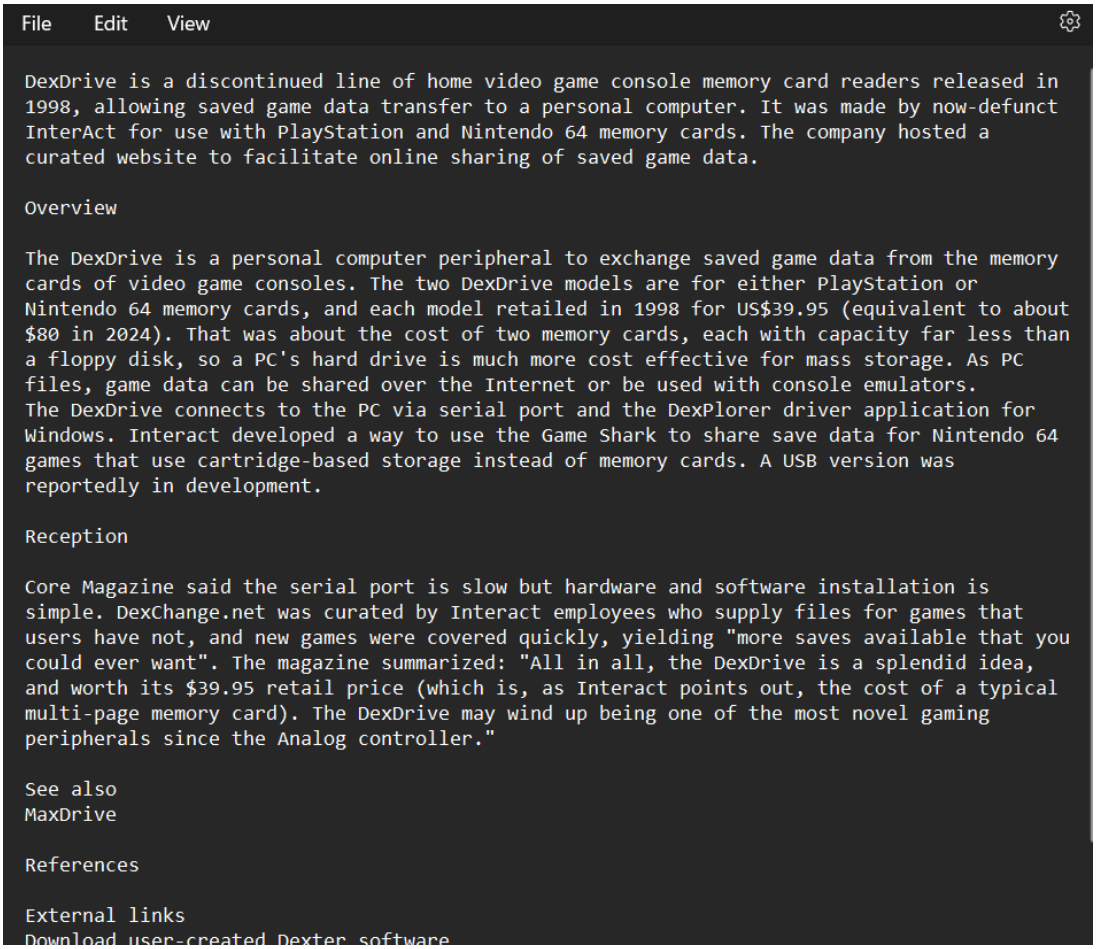

Рис. 2: Пример текста документа

2 Примеры существующих поисковых систем

Для анализа особенностей и ограничений современных поисковых систем были рассмотрены примеры поиска информации с использованием встроенного поиска Википедии и поисковой системы Google. В качестве запросов использовались термины, связанные с тематикой видеоигр.

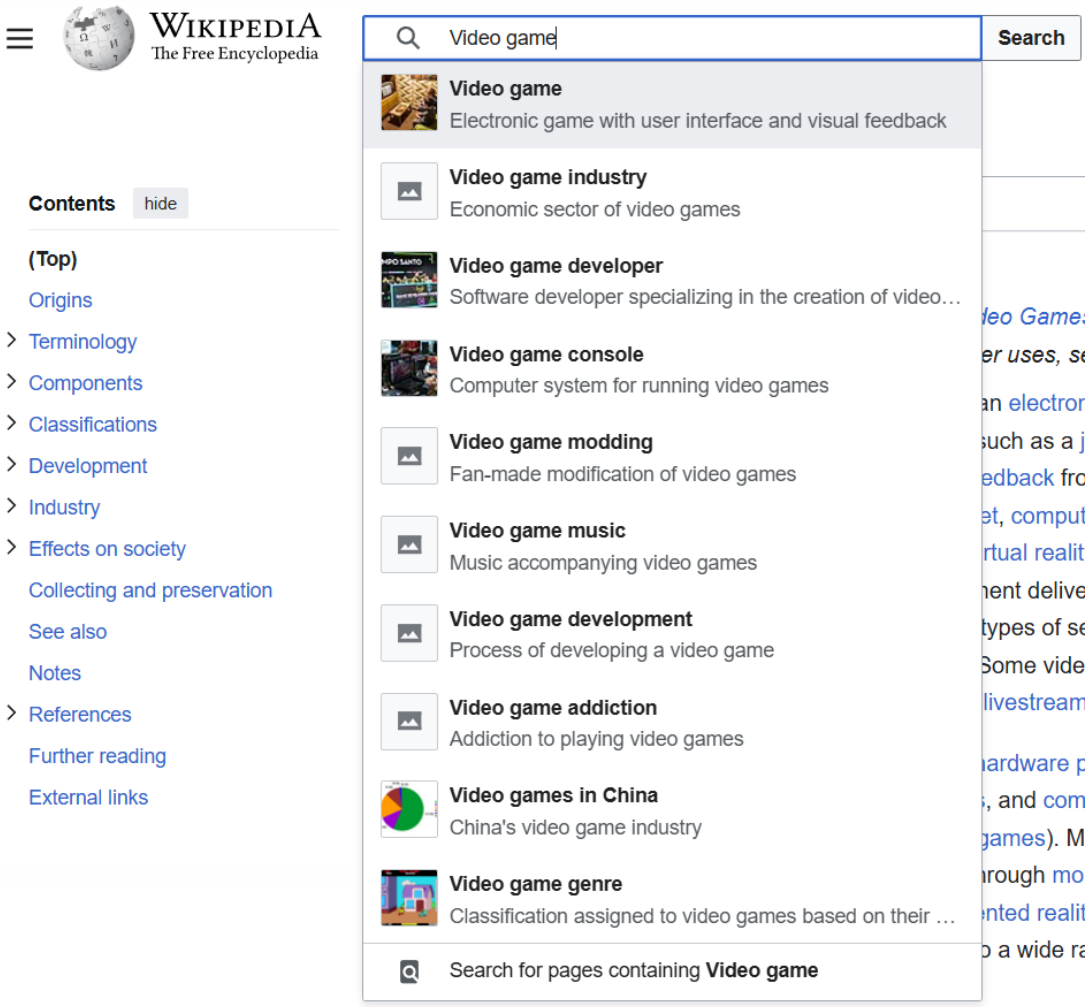

Рис. 3: Пример поиска в Википедии

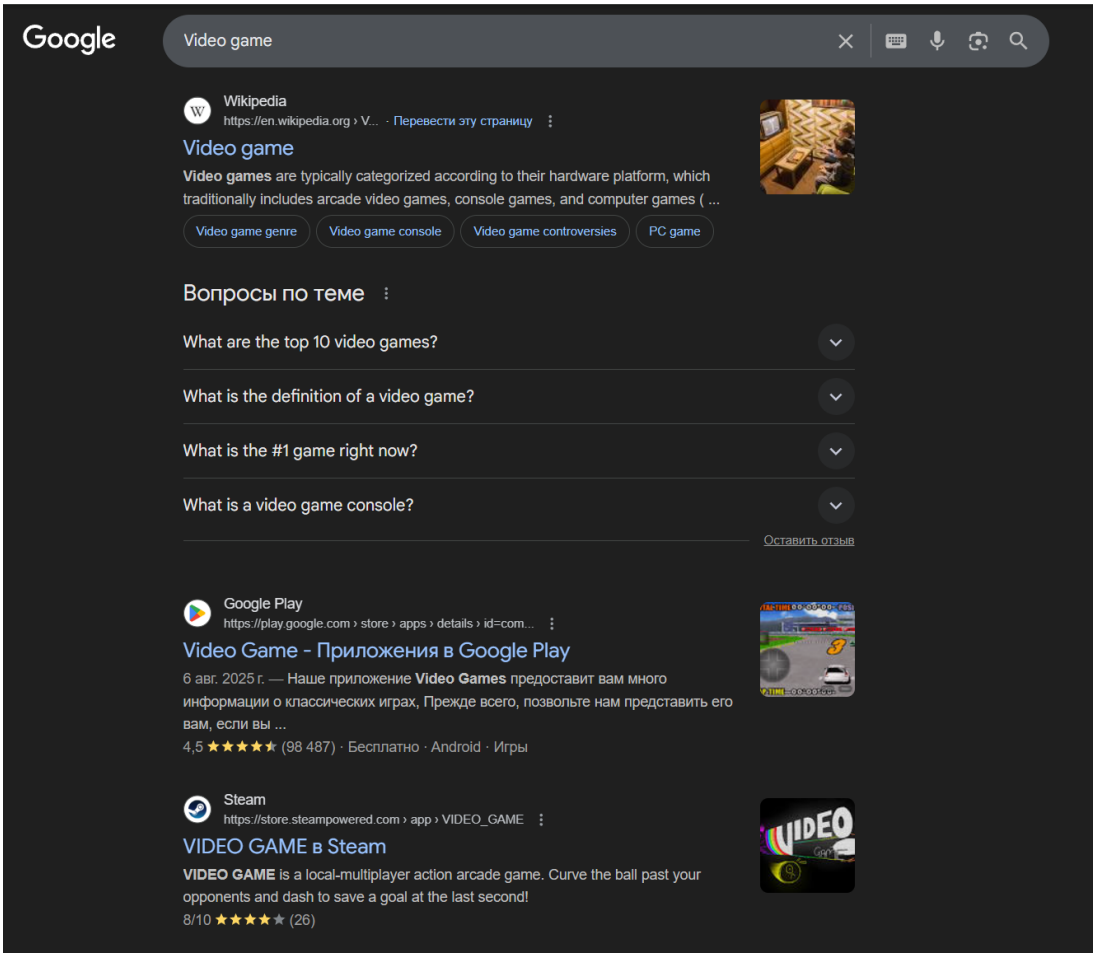

Рис. 4: Пример поиска в Google

Анализ полученных результатов показывает, что существующие поисковые системы ориентированы на массового пользователя и популярные источники. При этом поиск по узкоспециализированным запросам часто требует дополнительного уточнения, а структура выдачи может содержать нерелевантные результаты или служебную информацию.

3 Ключевые фрагменты кода

Основная логика скачивания корпуса находится в `download_wiki_corpus.py`. Ниже — обход категорий и батчевое получение текстов через API:

```
API_URL = "https://en.wikipedia.org/w/api.php"
ROOT_CATEGORY = "Category:Video games"
TARGET_DOCS = 30000
```

```
MAX_DEPTH = 6

def get_category_members(category_title, session):
    pages, subcats, cont = [], [], {}
    while True:
        data = api_get({...,"cmtitle": category_title, **cont}, session)
        for m in data.get("query", {}).get("categorymembers", []):
            title = m.get("title", "")
            if title.startswith("Category:"):
                subcats.append(title)
            else:
                pid = m.get("pageid")
                if isinstance(pid, int):
                    pages.append((pid, title))
        if "continue" not in data:
            break
        cont = data["continue"]
        time.sleep(REQUEST_PAUSE)
    return pages, subcats

def main():
    q = deque([(ROOT_CATEGORY, 0)])
    while q and doc_id < TARGET_DOCS:
        cat, depth = q.popleft()
        pages, subcats = get_category_members(cat, session)
        if depth < MAX_DEPTH:
            for sc in subcats:
                q.append((sc, depth + 1))
        #   plaintext-    corpus/
```

3 Лабораторная работа №2. Анализ статистических свойств корпуса

Для анализа статистических свойств корпуса был рассмотрен закон Ципфа, описывающий распределение частот слов в естественных языках. Перед выполнением анализа корпус был предварительно обработан. На этапе токенизации текст документов разбивался на последовательность токенов, при этом удалялись знаки пунктуации и приводился единый регистр. Далее применялся стемминг, позволяющий привести различные словоформы к общей основе и уменьшить размер словаря.

После нормализации текста для каждого уникального терма была подсчитана частота его вхождения во всём корпусе. Полученные данные были отсортированы по убыванию частоты, после чего каждому терму был присвоен ранг. На основе полученного распределения была построена зависимость частоты слова от его ранга.

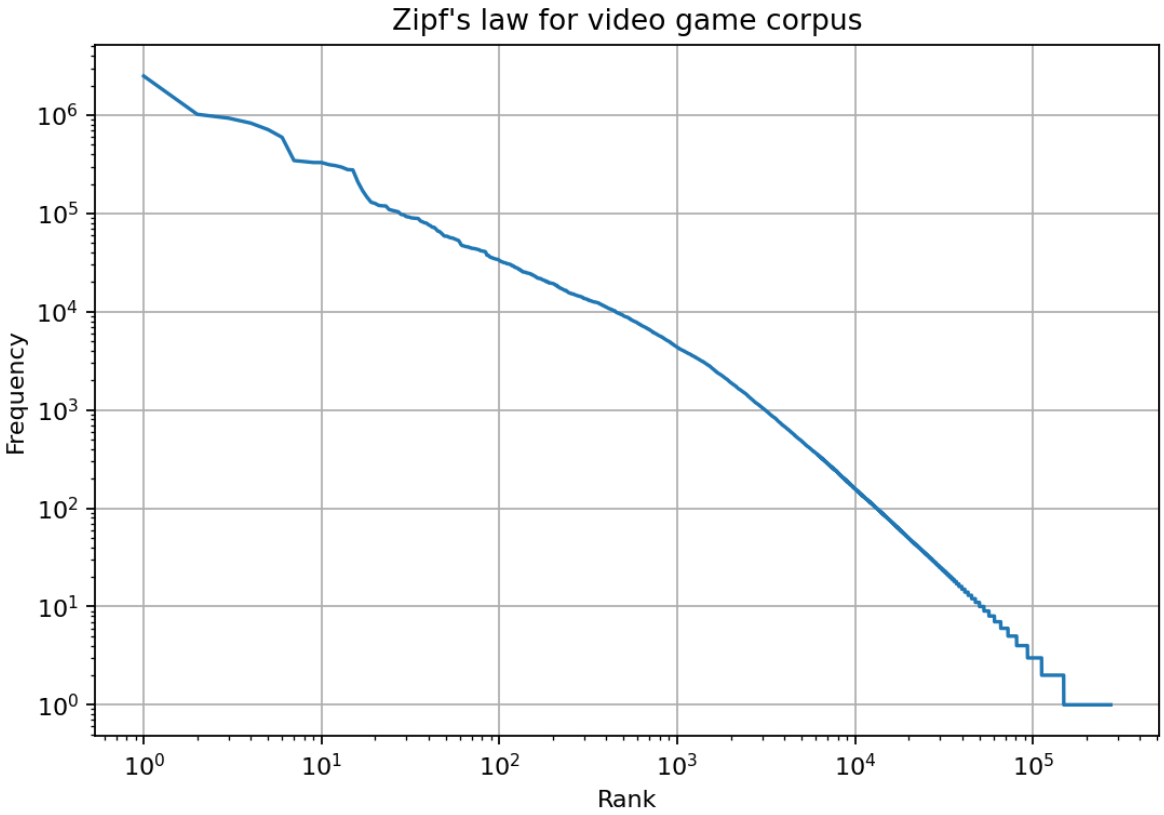

Рис. 5: Закон Ципфа для корпуса статей о видеоиграх

```
rank    word    freq
1       the     2502562
2       and     1020405
3       of      932727
4       to      830274
5       in      713531
6       game    595295
7       wa      345416
8       for     336695
9       as      330005
```

Анализ графика показывает, что распределение слов по частотам близко к линейному в логарифмических координатах, что соответствует закону Ципфа. Отклонения наблюдаются для наиболее частотных слов и в хвосте распределения, что характерно для реальных текстовых корпусов и подтверждает корректность этапов предварительной обработки текста.

1 Ключевые фрагменты кода

Подсчёт частот стемов (`zipf_freq.py`):

```
from collections import Counter
freq = Counter()
with open("stems.txt", "r", encoding="utf-8") as f:
    for line in f:
        w = line.strip()
        if w:
            freq[w] += 1
with open("freq.tsv", "w", encoding="utf-8") as out:
    out.write("rank\tword\tfreq\n")
    for rank, (word, count) in enumerate(freq.most_common(), start=1):
        out.write(f"{rank}\t{word}\t{count}\n")
```

Построение лог–лог графика распределения (`zipf_plot.py`):

```
import matplotlib.pyplot as plt
ranks, freqs = [], []
with open("freq.tsv", "r", encoding="utf-8") as f:
    next(f)
    for line in f:
        r, _, fr = line.strip().split("\t")
        ranks.append(int(r)); freqs.append(int(fr))
plt.loglog(ranks, freqs)
plt.xlabel("Rank"); plt.ylabel("Frequency")
plt.title("Zipf's law for video game corpus")
plt.savefig("zipf.png", dpi=150)
```

4 Лабораторная работа №3. Индексация корпуса документов

Для обеспечения быстрого поиска по корпусу документов был реализован булев инвертированный индекс. Инвертированный индекс сопоставляет каждому терму список идентификаторов документов, в которых данный терм встречается. Такой подход позволяет выполнять поиск по логическим условиям без полного перебора всех документов корпуса.

В процессе индексирования каждый документ обрабатывается независимо. Из документа извлекается набор уникальных термов, что позволяет избежать дублирования идентификатора документа в постинг-листе при многократном вхождении одного и того же слова. Для каждого терма формируется отсортированный список идентификаторов документов, в которых он присутствует. В дальнейшем данные списки используются для выполнения операций булевого поиска.

Индекс сохраняется на диске в виде двух основных файлов: словаря терминов и бинарного файла постинг-листов. В словаре для каждого терма хранится количество документов, в которых он встречается, а также смещение и длина соответствующего постинг-листа в бинарном файле. Такая организация позволяет загружать в память только необходимые данные при обработке поискового запроса и обеспечивает масштабируемость решения при увеличении размера корпуса. Дополнительно сохраняется общее количество документов корпуса, используемое для корректной реализации оператора отрицания.

```
>head index/dict.tsv
term    df      offset  len
0-0     8       0       32
0-0-7   1       32      4
0-00-713655-2   1       36      4
0-00-717558-2   1       40      4
0-00-720907-x   3       44      12
0-007-24622-6   2       56      8
0-02-935671-7   1       64      4
0-049-28039-2   1       68      4
0-06-083305-x   1       72      4

>wc -l index/dict.tsv
274105 index/dict.tsv

>ls -lh index/dict.tsv index/postings.bin index/maxdoc.txt
-rwxrwxrwx 1 user user 5.9M Dec 26 12:25 index/dict.tsv
```

```
-rwxrwxrwx 1 user user    6 Dec 26 12:25 index/maxdoc.txt
-rwxrwxrwx 1 user user  46M Dec 26 12:25 index/postings.bin

>cat index/maxdoc.txt
30000
```

1 Ключевые фрагменты кода

Сбор уникальных пар терм–документ и запись постинг-листов (`build_index.cpp`):

```cpp
struct Pair { std::string term; uint32_t doc; };

for (const auto& entry : fs::directory_iterator(stems_dir)) {
    if (p.extension() != ".stm") continue;
    uint32_t docId = parse_doc_id_from_filename(p);
    std::vector<std::string> terms;
    std::string w;
    while (read_line(in, w)) if (!w.empty()) terms.push_back(w);
    std::sort(terms.begin(), terms.end());
    terms.erase(std::unique(terms.begin(), terms.end()), terms.end());
    for (const auto& t : terms) pairs.push_back({t, docId});
}

std::sort(pairs.begin(), pairs.end(), [](auto& a, auto& b){
    return a.term == b.term ? a.doc < b.doc : a.term < b.term;
});

while (i < pairs.size()) {
    std::vector<uint32_t> docs;
    while (j < pairs.size() && pairs[j].term == term) {
        if (d != last) docs.push_back(d);
    }
    postings.write(reinterpret_cast<const char*>(docs.data()),
                   docs.size()*sizeof(uint32_t));
    dict << term << "\t" << docs.size() << "\t" << offset
         << "\t" << docs.size()*sizeof(uint32_t) << "\n";
}
```

5 Лабораторная работа №4. Реализация булевого поиска

Для демонстрации работы реализованной поисковой системы были выполнены несколько булевых запросов с использованием логических операторов `AND`, `OR` и `NOT`, а также скобок для явного задания приоритета операций. Поисковый запрос разбирается и преобразуется в последовательность операций над постинг-листами. Операции пересечения, объединения и разности выполняются над отсортированными списками идентификаторов документов.

Для корректной реализации оператора отрицания используется множество всех документов корпуса, размер которого определяется на этапе индексирования. В результате выполнения запроса пользователю возвращается упорядоченный список идентификаторов документов, удовлетворяющих заданным условиям.

```
Loaded terms: 274104
Universe docs: 1..30000
Enter queries. Ctrl+D to exit.

nintendo
RESULTS 8905
1
...
30000
END

10-year AND NOT 10-year-old
RESULTS 46
3
...
29891
END

10-year-old OR 10-year
RESULTS 81
3
...
28337
29891
END

(10-minute OR 10-yard) AND 10-year
```

```
RESULTS 0
END
```

1 Ключевые фрагменты кода

Парсер булевых выражений и выполнение операций (`boolean_search.cpp`):

```cpp
static std::vector<uint32_t> op_and(const std::vector<uint32_t>& a,
                                    const std::vector<uint32_t>& b) { ... }
static std::vector<uint32_t> op_or (const std::vector<uint32_t>& a,
                                    const std::vector<uint32_t>& b) { ... }
static std::vector<uint32_t> op_not(const std::vector<uint32_t>& a,
                                    const std::vector<uint32_t>& universe) { ... }

class Parser {
    std::vector<uint32_t> parse_or() {
        auto left = parse_and();
        while (match_op("OR")) {
            auto right = parse_and();
            left = op_or(left, right);
        }
        return left;
    }
    std::vector<uint32_t> parse_not() {
        if (match_op("NOT")) return op_not(parse_not(), universe);
        return parse_primary();
    }
    std::vector<uint32_t> parse_primary() {
        if (match("(")) { auto r = parse_or(); match(")"); return r; }
        return postings_for_term(t[pos++]);
    }
};
```

6 Лабораторная работа №5. Статистика работы системы

Для оценки эффективности реализованной системы была измерена производительность основных этапов обработки данных. В частности, было зафиксировано время построения индекса для полного корпуса.

```
>/usr/bin/time -p ./build_index --stems stems --out index

Processed docs: 500,pairs: 337955
Processed docs: 1000,pairs: 565153
...
Processed docs: 30000,pairs: 11974986
Index built.
Docs processed: 30000
maxDoc: 30000
Output: index/dict.tsv,postings.bin,maxdoc.txt
real 39.78
user 8.13
sys 3.67
```

7 Заключение

В ходе выполнения серии лабораторных работ по курсу «Информационный поиск» были изучены основные принципы построения поисковых систем, включая сбор и подготовку корпуса документов, анализ статистических свойств текста, индексирование и реализацию булевого поиска. Был сформирован корпус большого объёма, обеспечивающий репрезентативность результатов и корректность статистического анализа.

Реализованная поисковая система демонстрирует корректную работу логических операций поиска и позволяет выполнять поиск по корпусу документов без полного перебора данных. Архитектура решения обеспечивает воспроизводимость результатов и возможность повторного использования индекса. В дальнейшем систему можно расширить за счёт внедрения методов ранжирования документов, позиционного поиска или дополнительных способов анализа текстов.

Список литературы

[1] Маннинг, Рагхаван, Шютце *Введение в информационный поиск* — Издательский дом «Вильямс», 2011. Перевод с английского: доктор физ.-мат. наук Д. А. Клюшина — 528 с. (ISBN 978-5-8459-1623-4 (рус.))

[2] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. (ISBN 978-0521865715)

[3] MediaWiki API documentation. https://www.mediawiki.org/wiki/API:Main_page (дата обращения: 30.12.2025).