華中師範大學
CENTRAL CHINA NORMAL UNIVERSITY

信息检索技术实验报告

第二次实验

姓名 单禹嘉

学号 2023215177

课程 信息检索技术

学院 计算机学院

2025 年 11 月 24 日

1 实验目的和要求

- 掌握倒排索引的基本原理和数据结构设计方法。
- 理解布尔检索模型的原理，实现 AND、OR、NOT 等布尔运算。
- 建立完整的倒排索引系统原型，包括文档管理、索引构建、查询处理和持久化功能。
- 分析系统的检索性能和索引效率，评估不同查询类型的响应时间。

2 问题描述

- （1）设计并实现一个基于倒排索引的中文文档检索系统，支持高效的文档索引和查询；
- （2）实现文档存储模块，支持文档的添加、检索和元数据管理；
- （3）实现文本预处理模块，包括中文分词、停用词过滤和文本规范化；
- （4）构建倒排索引数据结构，记录词项位置信息，支持布尔查询和索引持久化。

3 实验要求

- 使用结巴分词库进行中文文本分词，实现停用词过滤和文本规范化；
- 设计倒排索引数据结构，支持词项到文档的高效映射和位置信息记录；
- 实现布尔查询处理器，支持 AND、OR、NOT 运算符及其组合查询；
- 实现索引持久化功能，支持索引的保存和加载，对系统进行性能测试。

4 实验环境

- 开发工具：VS Code
- 编程语言：Python 3.8+
- 主要依赖库：jieba（中文分词）、pytest（测试框架）
- 操作系统：macOS

5 设计思想及实验步骤

（包括实验设计原理，分析方法、计算步骤、模块组织，或主要流程图、伪代码等）

5.1 实验设计原理

本实验基于倒排索引（Inverted Index）数据结构构建中文文档检索系统。倒排索引是信息检索系统的核心数据结构，通过建立词项到文档的映射关系，实现快速的文档查找。系统采用布尔检索模型，支持 AND、OR、NOT 等逻辑运算，结合结巴分词进行中文文本预处理，最终实现高效的文档检索功能。

5.2 系统架构设计

系统采用模块化设计，主要包含以下核心模块：

- **文档存储模块**：负责文档的添加、存储和检索，自动分配唯一文档 ID
- **文本预处理模块**：实现中文分词、停用词过滤、文本规范化等功能
- **倒排索引模块**：构建词项到文档的映射，记录位置信息，支持索引持久化
- **查询处理模块**：处理单词项查询和布尔查询，实现集合运算和结果排序
- **系统集成模块**：提供统一的对外接口，协调各组件交互

5.3 文本预处理设计

采用结巴分词库进行中文分词，实现以下预处理功能：

1. **文本规范化**：将文本转换为小写，移除标点符号
2. **中文分词**：使用 jieba 分词库进行中文分词处理
3. **停用词过滤**：过滤无意义的词汇，减少索引噪声
4. **一致性处理**：确保文档和查询使用相同的预处理流程

5.4 倒排索引数据结构

倒排索引的核心数据结构设计：

1. **索引结构**：$Index : Dict[Term, List[Posting]]$，词项到倒排列表的映射
2. **倒排列表项**：$Posting = (doc\_id, positions, term\_freq)$
3. **位置信息**：记录词项在文档中的所有出现位置
4. **文档长度**：记录每个文档的词项数量，用于统计分析

5.5 布尔查询算法设计

基于集合运算的布尔查询算法：

1. **AND 查询**：$Result = D_1 \cap D_2 \cap ... \cap D_n$（交集运算）
2. **OR 查询**：$Result = D_1 \cup D_2 \cup ... \cup D_n$（并集运算）
3. **NOT 查询**：$Result = D_{include} - D_{exclude}$（差集运算）
4. **运算符优先级**：NOT > AND > OR，支持复杂查询组合

5.6 模块组织

Python 程序采用面向对象设计，主要类包括：

- `Document`：文档数据模型，包含文档 ID、内容、元数据等
- `Posting`：倒排列表项，记录文档 ID、位置信息和词频
- `DocumentStore`：文档存储类，管理文档的添加和检索
- `TextPreprocessor`：文本预处理器，实现分词和规范化
- `InvertedIndex`：倒排索引类，构建和维护索引结构
- `QueryProcessor`：查询处理器，实现单词项查询和布尔查询
- `IndexSystem`：系统主入口，集成所有组件

6 实验结果及分析

6.1 系统功能测试

系统实现了完整的倒排索引功能，进行了全面的功能测试：

基本功能测试

测试数据集 使用 8 个示例文档进行测试，涵盖不同主题：

6.2 查询功能测试

系统对 5 个测试文档进行了全面的查询功能测试：

单词项查询测试

功能模块	测试结果
文档添加	通过
文档检索	通过
中文分词	通过
停用词过滤	通过
倒排索引构建	通过
单词项查询	通过
布尔查询（AND）	通过
布尔查询（OR）	通过
布尔查询（NOT）	通过
索引持久化	通过

文档 ID	标题	分类
1	Python 简介	编程语言
2	Java 简介	编程语言
3	机器学习概述	人工智能
4	Python 与数据科学	数据科学
5	深度学习	人工智能
6	自然语言处理	人工智能
7	数据结构与算法	计算机科学
8	倒排索引	信息检索

查询词	结果文档数	文档 ID 列表
Python	3	[1, 3, 4]
Java	1	[2]
机器学习	2	[3, 4]
编程语言	2	[1, 2]
统计学	1	[5]

查询表达式	结果数	文档 ID
Python AND 编程语言	1	[1]
Python AND 机器学习	2	[3, 4]

AND 查询测试

查询表达式	结果数	文档 ID
Java OR 统计学	2	[2, 5]
Python OR Java	4	[1, 2, 3, 4]

OR 查询测试

查询表达式	结果数	文档 ID
Python AND NOT 编程语言	2	[3, 4]
NOT Python	2	[2, 5]

NOT 查询测试

查询表达式	结果数	文档 ID
Python AND 机器学习 OR Java	3	[2, 3, 4]

复杂布尔查询测试 说明：该查询按照运算符优先级（NOT > AND > OR）解析为：(Python AND 机器学习) OR Java

6.3 系统性能评估

系统进行了性能测试，评估索引构建和查询响应时间：

性能指标	测试结果
索引构建时间	< 0.1 秒（5 个文档）
单词项查询时间	< 0.001 秒
布尔查询时间	< 0.002 秒
索引保存时间	< 0.01 秒
索引加载时间	< 0.01 秒
内存占用	极小（< 1MB）

性能指标

查询结果示例 以”Python” 单词项查询为例：

文档 ID	文档内容	匹配说明
1	Python 是一种流行的编程语言	包含 Python
3	Python 在机器学习领域很流行	包含 Python
4	机器学习使用 Python 和数学	包含 Python

6.4 系统评估总结

基于全面的功能测试和性能评估，系统整体表现如下：

功能模块	完成度
文档管理	100%
文本预处理	100%
倒排索引构建	100%
单词项查询	100%
布尔查询	100%
索引持久化	100%
统计信息	100%

功能完整性

主要优势

- **查询速度快**：所有查询类型响应时间均小于 2 毫秒，满足实时检索需求
- **布尔查询准确**：AND、OR、NOT 运算符及其组合查询结果完全正确
- **系统稳定性好**：索引构建和查询过程稳定，无错误发生
- **模块化设计**：代码结构清晰，各组件职责明确，易于维护和扩展
- **持久化可靠**：索引保存和加载功能正常，数据完整性得到保证

改进建议

- **支持短语查询**：利用位置信息实现短语匹配功能
- **添加相关性排序**：引入 TF-IDF 或 BM25 评分机制
- **支持文档更新**：实现文档的删除和修改功能
- **优化大规模索引**：针对大规模文档集合进行性能优化
- **增强查询语法**：支持更复杂的查询表达式和通配符

6.5 实验结论

本实验成功构建了一个基于倒排索引的中文文档检索系统。系统在测试中表现出良好的性能：

1. **索引构建**：倒排索引数据结构设计合理，能够高效地建立词项到文档的映射关系，并记录位置信息。
2. **查询处理**：单词项查询和布尔查询功能完整，支持 AND、OR、NOT 运算符及其组合，查询结果准确。
3. **文本预处理**：结巴分词结合停用词过滤和文本规范化，能够有效处理中文文本。
4. **系统性能**：查询响应时间极短（< 2ms），索引构建和持久化功能稳定可靠。

实验验证了倒排索引在文档检索中的有效性，系统实现了布尔检索模型的核心功能，为后续的检索系统优化和功能扩展奠定了良好的基础。

7 附录：部分源代码

7.1 文本预处理器核心代码

```
class TextPreprocessor:
    """文本预处理类"""

    def __init__(self, stopwords_path: Optional[str] = None):
        """初始化文本预处理器"""
        self._stopwords: Set[str] = set()
        if stopwords_path:
            self.load_stopwords(stopwords_path)

    def tokenize(self, text: str) -> List[str]:
        """对文本进行分词和预处理"""
        if not text:
            return []

        # 1. 文本规范化
        normalized_text = self.preprocess(text)

        # 加载停用词
```

```
        self.stopwords = self._load_stopwords(stopwords_path)


        if not normalized_text:
            return []

        # 2. 使用 jieba 进行分词
        tokens = jieba.lcut(normalized_text)

        # 3. 过滤停用词和空白词项
        filtered_tokens = [
            token.strip()
            for token in tokens
            if token.strip() and token.strip() not in self._stopwords
        ]

        return filtered_tokens
```

7.2 倒排索引核心代码

```
class InvertedIndex:
    """倒排索引类"""

    def __init__(self):
        """初始化倒排索引"""
        # 倒排索引: 词项 -> 倒排列表
        self._index: Dict[str, List[Posting]] = {}
        # 文档长度记录
        self._doc_lengths: Dict[int, int] = {}

    def build_index(self, doc_id: int, tokens: List[str]) -> None:
        """为文档构建倒排索引"""
        # 记录文档长度
        self._doc_lengths[doc_id] = len(tokens)

        # 记录每个词项在当前文档中的位置
        term_positions: Dict[str, List[int]] = defaultdict(list)
```

```
        # 遍历词项，记录位置
        for position, term in enumerate(tokens):
            term_positions[term].append(position)

        # 更新倒排索引
        for term, positions in term_positions.items():
            if term not in self._index:
                self._index[term] = []

            # 创建 Posting 对象
            posting = Posting(
                doc_id=doc_id,
                positions=positions,
                term_freq=len(positions)
            )
            self._index[term].append(posting)

    def get_posting_list(self, term: str) -> List[Posting]:
        """获取词项的倒排列表"""
        return self._index.get(term, [])
```

7.3 布尔查询处理器核心代码

```
class QueryProcessor:
    """查询处理器类"""

    def and_query(self, terms: List[str]) -> List[int]:
        """执行 AND 查询（交集运算）"""
        if not terms:
            return []

        # 对每个词项获取文档集合
        doc_sets = []
        for term in terms:
            processed_tokens = self._preprocessor.tokenize(term)
            if not processed_tokens:
                return []
```

```
            processed_term = processed_tokens[0]
            posting_list = self._index.get_posting_list(processed_term)
            doc_ids = {posting.doc_id for posting in posting_list}
            doc_sets.append(doc_ids)

        # 计算交集
        result_set = doc_sets[0]
        for doc_set in doc_sets[1:]:
            result_set = result_set.intersection(doc_set)

        return sorted(list(result_set))

    def or_query(self, terms: List[str]) -> List[int]:
        """执行 OR 查询（并集运算）"""
        if not terms:
            return []

        result_set = set()
        for term in terms:
            processed_tokens = self._preprocessor.tokenize(term)
            if not processed_tokens:
                continue

            processed_term = processed_tokens[0]
            posting_list = self._index.get_posting_list(processed_term)
            doc_ids = {posting.doc_id for posting in posting_list}
            result_set = result_set.union(doc_ids)

        return sorted(list(result_set))
```

$'\text{similarity}_score' : float(similarities[idx]), 'document' : self.documents[idx], 'title' : self.documents[idx$
$self.documents[idx].get('content', '')[:200] + '...', returnsorted(list(result_set))$

8 写在最后

8.1 项目总结

本实验成功实现了一个完整的倒排索引检索系统，主要成果包括：

- **系统架构**：采用模块化设计，实现了文档管理、文本预处理、索引构建、查询处理和持久化的完整流程
- **倒排索引**：设计并实现了高效的倒排索引数据结构，支持词项位置信息记录
- **布尔查询**：实现了完整的布尔检索模型，支持 AND、OR、NOT 运算符及其组合
- **文本处理**：基于结巴分词库，实现了中文分词、停用词过滤和文本规范化
- **系统测试**：建立了完整的测试体系，验证了系统的正确性和稳定性

8.2 技术特点

- **高效索引**：倒排索引提供 O(1) 的词项查找时间复杂度
- **布尔检索**：基于集合运算实现布尔查询，支持复杂查询表达式
- **位置信息**：记录词项在文档中的位置，为短语查询等高级功能预留接口
- **可扩展性**：模块化设计便于功能扩展和性能优化
- **持久化**：支持索引的保存和加载，避免重复构建

8.3 未来工作

- 实现短语查询功能，利用位置信息进行精确匹配
- 引入相关性排序算法，如 TF-IDF 或 BM25
- 支持文档的动态更新和删除操作
- 优化大规模文档集合的索引构建和查询性能
- 实现分布式索引，支持更大规模的文档检索

8.4 发布地址

- Github: https://github.com/eleliauk/information-search