《现代信息检索技术》期末大作业

作业要求：

1. 在内存队英文文档集构建一个倒排索引，并采用

   A AND B 和 A OR B 形式的布尔查询进行处理，文档切分采用空格切分。

2. 采用 JAVA 或 C++语言实现。

3. 提交纸质文档（源码必须，其他文档可选）

   文档名称：《现代信息检索技术》期末大作业。

摘要：信息检索是从大规模非结构化数据的集合中找出满足用户信息需求资料的过程。本文介绍了基本倒排索引的结构及构建过程，通常倒排索引包含词典和倒排记录表两部分，为详细介绍整个倒排索引工作原理，项目中将词典和倒排记录表输出为一个索引文件，查询时可选择是否从外部读入索引文件进行查询。本文还介绍了布尔检索模型并对其进行实现，分析了如何通过线性时间复杂度的合并方法和简单的查询优化方法来进行高效检索。

关键字：信息检索 倒排索引 布尔查询 词项 词典

1. 词项－文档（Term-Doc）关联矩阵

词项-文档关联矩阵是表达词项与文档之间所具有的一种包含关系的概念模型。图 1.1 中每一列代表一个文档，每行代表一个词项，打勾位置表示文档中有相对应的词项。

单词-文档矩阵					
	文档1	文档2	文档3	文档4	文档5
词汇1	✓			✓	
词汇2		✓	✓		
词汇3				✓	
词汇4	✓				✓
词汇5		✓			
词汇6			✓		

图 1.1 词项－文档关联矩阵

- 从纵向即文档维度来看，每列代表文档包含了哪些单词，例如文档 1 包含了词汇 1 和词汇 4，而不包含其它单词。
- 从横向即单词这个维度来看，每行代表了哪些文档包含了某个单词。如对于词汇 1 来说，文档 1 和文档 4 中出现过单词 1，而其它文档不包含词汇 1。

搜索引擎中索引本质上是实现“词项-文档矩阵”的具体数据结构，可以通过不同的方式来实现上述概念模型，如“倒排索引”、“签名文件”、“后缀树”等方式。但是各项实验数据表明，“倒排索引”是实现词项到文档映射关系的最佳实现方式，后续主要介绍“倒排索引”的结构及构建过程。

2. 倒排索引基本概念

文档集合(Document Collection)：由若干文档构成的集合称之为文档集合。如海量的互联网网页或者大量的电子邮件都是文档集合的具体例子。

文档编号(Document ID)：在搜索引擎内部，会将文档集合内每个文档赋予一个唯一的内部编号，以此编号来作为该文档的唯一标识，这样方便内部处理，每个文档的内部编号即称之为“文档编号”，文中将会用 DocID 来表示文档编号。

单词编号(Word ID)：与文档编号类似，搜索引擎内部以唯一的编号来表征某个单词，单词编号可以作为某个单词的唯一表征。

倒排索引(Inverted Index)：倒排索引是实现“词项-文档矩阵”的一种具体存储形式，通过倒排索引，可以根据单词快速获取包含这个单词的文档列表。倒排索引主要由两个部分组成：“单词词典”和“倒排列表”。

倒排列表(Posting List)：倒排列表记载了出现过某个单词的所有文档的文档列表及单词在该文档中出现的位置信息，每条记录称为一个倒排项(Posting)。根据倒排列表，即可获知哪些文档包含某个单词。所有单词的倒排列表往往顺序地存储在磁盘的某个文件里，这个文件即被称之为倒排文件，倒排文件是存储倒排索引的物理文件。

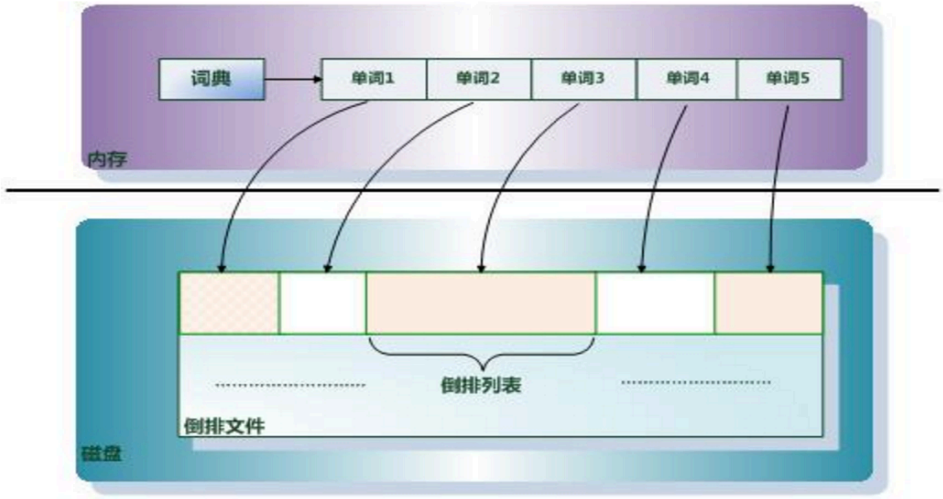

图 2.1 倒排索引示意图

3. 倒排索引的实现

倒排索引从逻辑结构和基本思路上来讲非常简单。下面我们通过具体实例来进行说明：

3.1 构建词项序列：<词项，文档 ID>二元组

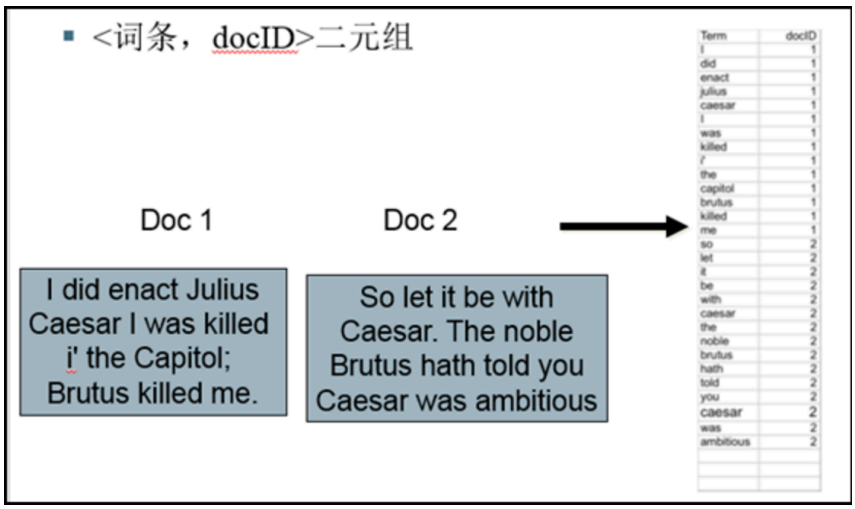

图 3.1 <词项，文档 ID>二元组

3.2 词典与倒排记录表

先按词项排序，然后每个词项按文档 ID 排序

图 3.2 排序后<词项，文档 ID>二元组

a) 词典 倒排记录表

i. 某个词项在单篇文档中的多次出现会被合并

ii. 分别构建词典和倒排记录表

iii. 添加每个词项出现的文档频率(doc frequency, DF)

iv. 〈词项，文档频率〉 ——>>〈倒排列表〉

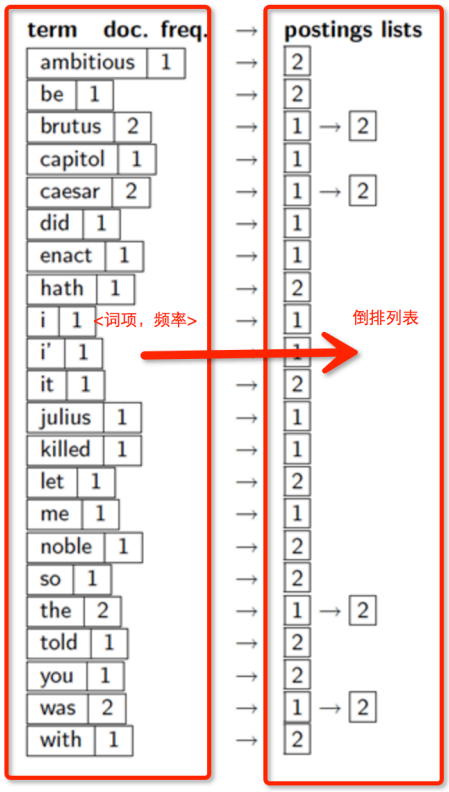

图 3.3 词典与倒排记录表关系图

3.3 词典与倒排记录表实现

```
TreeMap<String, TreeMap<Integer, int[]>> tmpmap = new TreeMap<String, TreeMap<Integer, int[]>>();
TreeMap<Integer, int[]> docsMap;                    // 文档频率 所在文档集的数组
```

图 3.4 词典与倒排记录表数据结构

- **词典及倒排列表数据结构简介：**

  TreeMap<String , TreeMap<Integer, int[]>> tmpmap;
  词项，倒排记录表（词项频率，文档 ID）

  TreeMap<Integer , int[]> docsMap;
  倒排记录表（词项频率， 文档 ID）

- **创建倒排索引过程：**
  - 1. 从文档集中读入文档
  - 2. 提取出文档各词项
  - 3. 构建<词项，倒排列表>二元组
  - 4. 更新文档频率及倒排列表
  - 5. 输出词典及倒排列表为文件形式（Dictionary.txt）

```java
public TreeMap<String, TreeMap<Integer, int[]>> Create() throws Exception {
    // map中key为单词项, value 为新的TreeMap<词频, 所出现的位置>
    TreeMap<String, TreeMap<Integer, int[]>> map = new TreeMap<String, TreeMap<Integer, int[]>>();
    TreeMap<Integer, int[]> Smap;
    int[] arr;
    try {
        // 这里假设文档集有5个文档
        for (int DocId = 1; DocId <= 5; DocId++) {
            String line = "";
            BufferedReader br = new BufferedReader(new FileReader("docs/doc" + DocId));
            line = br.readLine();
            while (line != null) {
                String[] text = line.split("[^A-Za-z]+");
                for (int i = 0; i < text.length; i++) {
                    text[i] = text[i].toLowerCase();
                    if (!map.containsKey(text[i])) {
                        Smap = new TreeMap<Integer, int[]>();
                        arr = new int[10];
                        arr[0] = DocId;
                        Smap.put(1, arr);
                        map.put(text[i], Smap);
                    } else {
                        int count = 0;
                        Smap = new TreeMap<Integer, int[]>();
                        int a = this.getkey(map.get(text[i]));
                        int[] tmp = this.getvalue(map.get(text[i]));
                        for (int j = 0; j < tmp.length; j++) {
                            if (tmp[j] == DocId) {
                                count++;
                            }
                        }
                        if (count == 0) {
                            a++;
                            tmp[a - 1] = DocId;
                            Smap.put(a, tmp);
                            map.put(text[i], Smap);
                        }
                    }
                }
                line = br.readLine();
            }
```

创建倒排索引

1.从文档集中读入文档

// 读取一行

// 正则表达式匹配,分离中其中以字母开头的词(即英文单词)

2.提取出文档各词项

// 如果词典中还没有当前单词,则加入其中

// 如果词典中已经有当前单词,则增加频率

3.新建倒排列表

// 获取词项的频率

// 获取词项出现的位置

4.更新文档频率

图 3.5 词典与倒排记录表实现核心代码

- **字典及倒排列表输出：**
  - 1. 遍历 TreeMap map，输出所有词项
  - 2. 遍历 TreeMap Smap，输出所有词项对应的文档 ID
  - 3. 生成字典及倒排记录表文件 Dictionary.txt

```java
// 字典,倒排列表输出
BufferedWriter out = new BufferedWriter(new FileWriter("Dictionary.txt"));
Iterator it = map.keySet().iterator();
while (it.hasNext()) {
    String tmp = (String) it.next();    // 获取词项
    TreeMap<Integer, int[]> DocFeq = map.get(tmp);
    int[] temp = getvalue(DocFeq);
    int s = getkey(DocFeq);
    out.write(String.format("%20s" + " " +"%5s",tmp, s));
    for (int t = 0; t < temp.length; t++) {
        if (temp[t] != 0)
            out.write(String.format(" " + "%4s",temp[t]));
    }
    out.newLine();
}
out.close();
return map;
```

1.遍历TreeMap，输出所有词项

2.遍历TreeMap，输出对应文档ID

字典 倒排列表输出

图 3.6 词典与倒排记录表输出核心代码

- **输出文件格式简介**

```
bareness        1       3
      be        2       1       2
    bear        1       1
  beauty        4       1       2       3       4
     bed        1       5
```

图 3.7 输出文件格式（词项 频率 文档 ID[]）

4. 布尔查询的处理

已有如图 4.1 所示倒排索引表，利用索引来处理查询

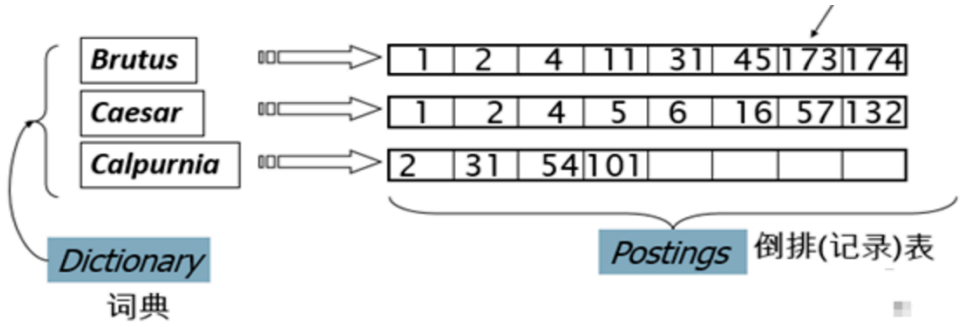

图 4.1 布尔查询倒排索引表实例

4.1 AND 查询的处理：

考虑如下查询（从简单的布尔表达式入手）：

- Brutus AND Caesar
  - 在词典中定位 Brutus
  - 返回对应倒排记录表(对应的 docID)
  - 在词典中定位 Caesar
  - 再返回对应倒排记录表
  - 合并(Merge)两个倒排记录表，即求交集

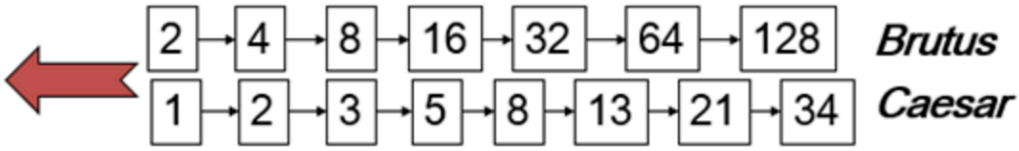

图 4.2 AND 查询倒排记录表

- **合并过程**

每个倒排记录表都有一个定位指针，两个指针同时从前往后扫描， 每次比较当前指针对应倒排记录，然后移动某个或两个指针。合并时间为两个表长之和的线性时间。

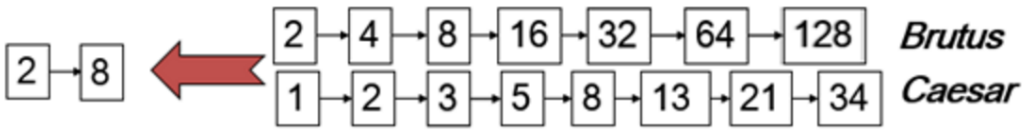

图 4.3 AND 查询倒排记录表合并过程

- 假定表长分别为 x 和 y，那么上述合并算法的复杂度为 O(x+y)
- 关键原因：倒排记录表按照 docID 排序
- 上述合并算法的伪代码：

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

图 4.4 AND 查询倒排记录表合并伪代码

4.2 OR 查询的处理：

OR 查询的处理同 AND 查询，只需要处理两个倒排记录表的并集。

4. 3 NOT 查询的处理

NOT 查询的处理同 AND 查询，只需将两个倒排记录表相减。

4.4 布尔查询的代码实现

```
public int[] AndReslut(String word1, String word2, TreeMap<String, TreeMap<Integer, int[]>> map) {
    List<Integer> list = new LinkedList<Integer>();
    int[] tmp1 = new int[0];
    int[] tmp2 = new int[0];
    if (map.get(word1) != null) {
        tmp1 = getvalue(word1, map);
    } else
        System.out.println("第一个单词不存在！");

    if (map.get(word2) != null) {
        tmp2 = getvalue(word2, map);
    } else
        System.out.println("第二个单词不存在！");

    // tmp1 和 tmp2都是已经排序的,可以优化
    for (int i = 0, j = 0, k = 0; i < tmp1.length + tmp2.length; i++) {
        if (j < tmp1.length && k < tmp2.length) {
            if (tmp1[j] < tmp2[k]) {
                j++;
            } else if (tmp1[j] > tmp2[k]) {
                k++;
            } else {
                list.add(tmp1[j]);
                j++;
                k++;
            }
        } else break;
    }

    if (list.isEmpty()) {
        int[] a = {-1};      // 满足此条件的文档不存在
        return a;
    }
    int[] tmp = new int[list.size()];
    for (int m = 0; m < tmp.length; m++)
        tmp[m] = list.get(m);
    return tmp;
}
```

AND查询

1.获取第一个单词的（word1）倒排记录表

2.获取第二个单词的（word2）倒排记录表

3.求倒排记录表tmp1和tmp2的交集

4.输出交集，文档ID数组 int[]

图 4.5 AND 查询核心代码

- AND 查询实现
  - 1. 获取第一个单词（word1）的倒排记录表
  - 2. 获取第二个单词（word2）的倒排记录表
  - 3. 求倒排记录表 tmp1 和 tmp2 的交集
  - 4. 返回文档，文档 ID 数组 int[] tmp

- OR 查询实现
  - 1. 获取第一个单词（word1）的倒排列表
  - 2. 获取第二个单词（word2）的倒排列表
  - 3. 返回两个倒排列表的并集

```
public int[] OrReslut(String word1, String word2, TreeMap<String, TreeMap<Integer, int[]>> map) {
    Set<Integer> set = new TreeSet<Integer>();
    if (map.get(word1) != null) {
        int tmp1[] = getvalue(word1, map);
        for (Integer i : tmp1)
            set.add(i);
    } else
        System.out.println("第一个单词不存在！");

    if (map.get(word2) != null) {
        int tmp2[] = getvalue(word2, map);
        for (Integer i : tmp2)
            set.add(i);
    } else
        System.out.println("第二个单词不存在！");

    if (set.isEmpty()) {
        int a[] = {-1};
        return a;
    }
    int[] tmp = new int[set.size()];
    int i = 0;
    Iterator<Integer> it = set.iterator();
    while (it.hasNext()) {
        tmp[i++] = it.next();
    }
    return tmp;
}
```

OR查询

1.获取第一个单词（word1）的倒排列表

2.获取第二个单词（word2）的倒排列表

3.返回两个倒排列表的并集

图 4.6 OR 查询核心代码

- 查询整体实现
  - 1. 单词项查询，直接调用查询函数
  - 2. 判断是否为布尔查询，执行布尔查询

```
if (tmp[1].equals("and")) {
    int[] temp1 = AndReslut(tmp[0], tmp[2], map);
    if (temp1[0] == -1) {
        System.out.println("不存在!");
    } else {
        OutPut(temp1);
    }
}
if (tmp[1].equals("or")) {
    int[] temp2 = OrReslut(tmp[0], tmp[2], map);
    if (temp2[0] == -1)
        System.out.println("不存在!");
    else {
        OutPut(temp2);
    }
}
```

图 4.6 查询核心代码

5. 项目运行介绍

- **整体主函数部分**
  - 1. 词典 倒排列表数据结构定义
  - 2. 选择是否使用外部索引文件
  - 3. 若选择读取外部索引，则读取索引文件
  - 4. 进行查询处理

```
public static void main(final String[] args) throws Exception {
    TreeMap<String, TreeMap<Integer, int[]>> tmpmap = new TreeMap<String, TreeMap<Integer, int[]>>();
    TreeMap<Integer, int[]> docsMap;                                    // 文档频率 所在文档集的数组
    System.out.println("1.是否需要重新生成外部字典? 是(Y), 否(N)");
    Scanner command = new Scanner(System.in);                                          1.词典 倒排列表数据结构
    String command1 = command.next();
    // 1.重新生成字典,并直接载入内存中
    if (command1.toLowerCase().equals("y")) {                                          2.选择是否使用外部索引
        Index createIndex = new Index();
        tmpmap = createIndex.Create();
    } else {
        // 2.从外部字典载入
        BufferedReader bufferedReader = new BufferedReader(new FileReader("Dictionary.txt"));
        String line = bufferedReader.readLine();
        while (line != null) {
            line = line.replaceAll("\\s+", " ");                        // 处理字符串中的多空格问题
            String temp[] = line.trim().split(" ");                     // 已空格隔开
            String word = temp[0];                                      // 提取出单词
            int docFre = Integer.parseInt(temp[1]);                     // 提取词项的出现频率
            int[] docs = new int[temp.length - 2];
            for (int i = 2; i < temp.length; i++) {
                docs[i - 2] = Integer.parseInt(temp[i]);
            }                                                                          3.读取外部索引文件
            docsMap = new TreeMap<Integer, int[]>();
            docsMap.put(docFre, docs);
            tmpmap.put(word, docsMap);
            line = bufferedReader.readLine();
        }
    }
    Inquire search = new Inquire();
    search.inquire(tmpmap);                                                            4.处理查询
}
```

图 5.1 整体主函数部分核心代码块

- **程序运行介绍：**
  - 1. 从索引文件读入
  - 2. AND 查询验证
  - 3. OR 查询验证
  - 4. NOT 查询验证

```
19          be      2    1    2
20        bear      1    1
21      beauty      4    1    2    3    4
22         bed      1    5                                   倒排索引文件
23      behold      1    5
24       being      1    2
25      bereft      1    3
26     besiege      1    2
27       birds      1    5
28       black      1    5
Run:  t   t   t   main   main   main   main
/Library/Java/JavaVirtualMachines/jdk1.8.0_73.jdk/Contents/Ho
1.是否需要重新生成外部字典? 是(Y), 否(N)                          1.从索引文件读入
N
2.请输入查询词项(支持布尔查询): 退出(E)
    查询格式: word1 and/or word2
be and bear                                                    2.AND查询验证
1
2.请输入查询词项(支持布尔查询): 退出(E)
    查询格式: word1 and/or word2
bear or behold                                                 3.OR查询验证
1  5
```

图 5.2 程序运行结果图

6. 作业总结

在搜索引擎实际的应用之中，有时需要按照关键字的某些值查找记录，所以我们按照关键字（本文中是按词项）建立索引，这个索引即倒排索引，而带有倒排索引的文件即倒排索引文件（倒排文件），通过其可以实现文档的快速检索。采用倒排索引优点主要在于:在处理复杂的多关键字查询时，可在倒排列表中先完成查询的交、并等逻辑运算，得到结果后再对记录进行存取，这样不必对每个记录随机存取，把对记录的查询转换为地址集合的运算，从而提高查找速度。然而通常布尔检索有如下缺点：（1）布尔查询构建优化复杂，不适合普通用户。如果构建不当，检索结果将出现过多或者过少的情况。（2）没有充分利用词项的频率信息。（3）检索过程中，不能对检索结果进行排序。

参考文献：

（1）[信息检索] 第一讲 布尔检索 Boolean Retrieval
http://www.cnblogs.com/AndyJee/p/3480273.html
（2）搜索引擎-倒排索引基础知识
http://blog.csdn.net/hguisu/article/details/7962350
（3）大数据处理--倒排索引
http://jeyke.iteye.com/blog/2086297
（4）《信息检索导论》Christopher D.Manning/Hinrich Schütze/PrabhakarRaghavan
出版社: 人民邮电出版社 第一章 第二章 第三章
（5）《基于倒排索引的全文检索技术研究》 作者：刘兴宇
华中科技大学计算机学院
（6）一种高效的倒排索引存储结构 作者：邓攀；刘功申
上海交通大学信息安全工程学院