****

课程设计报告书

**题目：《科学文献管理系统》**

**学 院 计算机科学与工程学院**

**专 业 计算机科学与技术2班**

**学生姓名 叶劲亨**

**学生学号 201830582180**

**指导教师 吕建明**

**课程编号**

**课程学分**

**起始日期 2020/01**

|  |  |
| --- | --- |
| 教  师  评  语 | 教师签名：  日期： |
| 成  绩  评  定 |  |
| 备  注 |  |

# 《科学文献管理系统》

## 选题背景

现在是一个大数据的时代，随着信息技术的发展和网络经济的快速进步，所以会有很多科学文献的出现，同时也就有各种各样的数据需要处理，就需要通过多种方式的资料整合来查询，科研工作者的日常工作离不开查阅科学文献，并对其中的信息进行分析、筛选、挖掘和管理。我们为科研工作者设计一个管理系统，提高科学文献的管理效率。

## 方案论证(设计理念)

### 需求分析

1. 基本搜索功能。输入作者名，能展示该作者发表的所有论文信息。输入完整的论文的题目，能展示该论文的其他相关信息 Hashtable
2. 相关搜索。输入作者名，能展示于该作者有合作关系的其他所以作者。
3. 作者统计功能。输出写文章最多的前100名作者。
4. 热点分析功能。分析每一年发表的文章中，题目所包含的单词中，出现频率排名前10的关键词。
5. 部分匹配搜索功能。给定若干个关键字，能快速搜索到题目中包含该关键字的文章信息
6. 聚团分析。作者之间的合作关系可以看成是一个图，每个作者对应一个顶点，任两个作者之间如果存在合作关系，则在两个顶点之间建立连边。这个图中的每一个完全子图我们称为一个聚团（所谓完全子图指的是该子图的任意顶点都和该子图的其他顶点有连边，完全子图的顶点个数称为该完全子图的阶数），请统计整个图中各阶完全子图的个数。
7. 可视化显示。通过图形化界面，展示作者之间合作关系图及其相关文章信息。

### 技术难点

为实现以上需求功能，我们决定使用Java语言实现基本功能，使用Web技术实现查询的可视化操作。其中，存在几个重点和难点：

1. 设计高效且准确的算法来读取超大XML原始文件，并将原数据文件中每一条记录以索引方式记录在索引文件中；
2. 要设计合适的数据结构来描述并存储所需查询的信息，以满足基础的查询功能；
3. 要设计合适的算法来实现快速、精确的查找，以及实现模糊查找，并优化查询速率；
4. 要设计合适的算法来提取出每一年内出现频率最高的十个关键词，其中需要注意关键词的提取需要有意义，不能出现类似于“for”,“of”之类的词；
5. 要设计合适的算法来实现聚团分析，统计一个超过200万节点的各阶完全子图的数量；
6. 要实现各个功能的可视化操作页面，并且实现页面的美观、实用，以及整套页面的完整逻辑。

### 技术选型

为实现以上的功能，我们决定采用以下技术选型：

* + 1. 采用仿Sax算法来高效读取超大XML原始文件。

比较主流高效的dom4j读取XML文件方式并不适用于读取大文件的xml，而常规Sax方式读取文件，则较难从读取的信息中抽取出我们所需要的的信息（该条记录所在的位置偏移量）。所以我们模仿Sax算法思路自己设计算法来读取原始XML文件。相比于sax方法所需要考虑到的情况更少，代码更加简单，对于此文件的适应程度更好，同时对于文件进行读取访问的次数也更少。

* + 1. 采用索引来实现记录的快速搜索

将每一条记录以key-value对形式存放在索引文件中，其中key是作者名或者标题名，value是在原数据文件（dblp.xml）中该记录的存放位置，索引文件最终以csv格式进行存储。由于索引具有高划分度、有序性、不可重复性三大特性，我们可以通过二分查找法来快速定位索引的位置，查询效率非常之高，可以快速通过title、author在索引文件中查找到相应的文献记录的地址，随后在原数据文件进行追踪，从而获取相应的文献记录。CSV文件有诸多优势，用来做数据存储容量比XML小, 功能比TXT强, 支持EXCEL直接查看，而且在java中读取csv文件非常方便高效。

* + 1. 部分搜索匹配

这部分的是实现由两种方案，一种采用正则表达式进行部分匹配，另外一种是全文检索。（匹配的情况是匹配多个完整的单词）。正则表达式部分匹配搜索(已经集成在项目中)是字符串处理中常用的方案，在项目中，我们采用先行断言+懒惰匹配+\b的方式，简洁而有效但速度不快。全文检索法是采用空间换取时间的方案（由于tomcat的问题，未能集成在项目中，可以在两秒内查询出两个单词组成的查询条件的结果），查询前会先将查询的内容（所有title组成的列表）提取出来组成文档（正文）,对文档进行切分词组成索引（目录）,索引和文档有关联关系，查询的时候先查询索引，通过索引找文档的这个过程叫做全文检索。

* + 1. 聚团分析

参考两篇论文《一个复杂网络中完全子图的搜索算法》、《复杂网络中的完全子图搜索算法研究》，并在其算法思想的基础上重新设计合适的统计算法。重新设计的算法流程简单易于理解，而且将时间复杂度降至为θ（n），可以在15分钟左右的时间统计完250万个节点的网络图的各阶完全子图数量。

* + 1. 使用分词实现热点分析功能

采用hashmap来记录每年的年度热词信息，包括年，该年热词。并存储在txt文档中。采用StringTokenizer来对句子进行划分，并且对于重复率较高的无意义词汇进行筛选处理。这样的方法效率较高，而且代码的复杂程度较低。采用hashmap来记录每年的年度热词信息，包括年，该年热词。并存储在txt文档中。以便于之后的可视化显示。

* + 1. Javaweb相关技术

对于图形界面的部分，我们考虑过采用Swing（java的一个工具包）进行图形用户界面的操作，但是想要设计出界面友好的图形用户界面会比较难。

因此我们考虑将可视化的部分部署在浏览器，数据来源自服务器，即传统的B-S架构模式。因此我们采用三层架构-SpringMVC来实现javaweb，实现javaweb的目的是便于借助网页进行可视化，在Web层面进行可视化和实现对用户操作的处理会比在系统层面的普通的GUI（图形用户界面）更加具备扩展性，例如关系图的绘制可以采用Echarts实现，加载查询结果的时候可以给出进度条提示，输入的查询参数不符合要求会给出错误提示以及大量数据的加载可以用ajax配合分页进行优化等等，大大减少了技术人员的开发量和提高了用户的使用体验。三层架构包括控制层，业务逻辑层，数据访问层。对于一个简单的应用程序来说，代码量不是很多的情况下，一层结构或二层结构开发完全够用，没有必要将其复杂化，如果对一个复杂的大型系统，设计为一层结构或二层结构开发，那么这样的设计存在很严重缺陷。

分层开发其实是为大型系统服务的。在开发过程中，初级程序人员出现相似的功能经常复制代码，那么同样的代码写那么多次，不但使程序变得冗长，更不利于维护，一个小小的修改或许会涉及很多页面，经常导致异常的产生使程序不能正常运行。最主要的面向对象的思想没有得到丝毫的体现，打着面向对象的幌子却依然走着面向过程的道路。

意识到这样的问题，程序开发人员开始将程序中一些公用的处理程序写成公共方法，封装在类中，供其它程序调用。例如写一个数据操作类，对数据操作进行合理封装，在数据库操作过程中，只要类中的相应方法（数据添加、修改、查询等）可以完成特定的数据操作，这就是数据访问层，不用每次操作数据库时都写那些重复性 的数据库操作代码。在新的应用开发中，数据访问层可以直接拿来用。面向对象的三大特性之一的封装性在这里得到了很好的体现。现在找到了面向对象的感觉，代码量较以前有了很大的减少，而且修改的时候也比较方便，也实现了代码的重用性。

controller层负责具体的业务模块流程的控制，在此层要调用service层的接口来控制业务流程，控制的配置也同样是在Spring的配置文件里进行，针对具体的业务流程，会有不同的控制器。我们具体的设计过程可以将流程进行抽象归纳，设计出可以重复利用的子单元流程模块。这样不仅使程序结构变得清晰，也大大减少了代码量。

service层主要负责业务模块的应用逻辑应用设计。同样是首先设计接口，再设计其实现类，接着再Spring的配置文件中配置其实现的关联。这样我们就可以在应用中调用service接口来进行业务处理。service层的业务实体，具体要调用已经定义的数据操作接口，封装service层业务逻辑有利于通用的业务逻辑的独立性和重复利用性。程序显得非常简洁。

view层与控制层结合比较紧密，需要二者结合起来协同开发。view层主要负责前台jsp页面的显示。

三层架构降低了层与层之间的依赖，层与层之间实现了解耦（把你的代码组织成最小组织单位（模块），并限制他们之间的交互）。各层的代码也具备复用性。

## 采用Echart生成作者合作关系图

此方案较为成熟，同时可复用性相较于个人所设计的关系图更高。不容易出现数据重叠的情况。不过存在对于电脑性能一定程度上的要求，如果生成的关系图的节点数量过多，会导致界面出现一定程度上的卡顿。经测试，在50个节点以上时，如果只使用集成显卡进行页面处理，会出现一定程度上的卡顿。

## 过程论述

**首先说明一下本人在该项目中的主要负责工作：**

1. **建立索引系统；**
2. **完成基本搜索功能（包括根据作者名搜索作者发表的所有文章记录、根据标题名称搜索文章记录）；**
3. **完成相关搜索功能（输入作者名，能展示于该作者有合作关系的其他所以作者）；**
4. **实现作者统计功能，统计发表文章数排名前100的作者；**
5. **实现页面可视化：SpringMVC环境搭建以及大部分网页设计；**

### 数据组织方式

* + 1. Author类的定义

Author类的定义如下。

|  |  |  |
| --- | --- | --- |
| 属性类型 | 属性名 | 描述 |
| String | name | 作者名 |
| List<Long> | locations | 索引位置集合 |
| List<String> | records | 包含此作者的记录集合 |
| Map<String, List<String>> | collaborators | 作者的合作者映射，key是共同创作的文章（论文）标题，value是这篇文章下的合作者名字 |

**表1 Author类属性表**

**P.S. 索引位置集合locations中索引的意义**

**为了达到通过作者名称，快速定位到原数据文件（dblp.xml）中与该作者相关记录的需要，该项目的解决方案是：**

**首先在磁盘中维护多个“有序”的索引文件，每个索引文件中的一行相当于一条索引记录，格式如下：**

**"Bert F. Lenting", 1197754108, 2197464017**

**其中，双引号包围起来的是作者名称，后面是原数据文件（dblp.xml）中与该作者相关记录的地址，因此，Bert F. Lenting这个作者发表了两篇文章，文章的地址分别是1197754108和2197464017。**

**值得一提的是，索引文件中记录的存放顺序符合字典有序性，这个规则有利于后续的二分查找相应索引记录。索引的更详细介绍将在后续展开。**

* + 1. Article类的定义

Article类定义如下。

|  |  |  |
| --- | --- | --- |
| 属性类型 | 属性名 | 描述 |
| String | title | 文章标题 |
| List<Long> | locations | 索引位置集合 |
| List<String> | records | 文章记录集合 |

**表2 Article类属性表**

**P.S. 为什么此处的文章记录是一个集合？**

在dblp.xml原数据文件中，含有许多同名标题的文章记录（标题为“Home Page”的文章记录就有多达几百万个），为了不丢失数据，这里使用List集合存储。

### 索引创建方式详解

该项目的核心在于索引文件的设计、创建与维护，通过索引，我们就可以以毫秒级速度（后续会有专门测试）查询作者和标题记录。该过程完全由本人实现并测试，测试结果表明设计十分成功。该项目中的索引具有三大特性。

* + 1. 索引的三大特性

### 高划分度

试想如果要存放关于作者记录的索引，你是希望用一个文件来存放所有作者索引呢，还是用多个文件来存放呢？这其实就是索引的物理存储问题，一个好的物理存储设计能够带来性能上的显著提升。毫无疑问，一个好的设计应该是把作者索引分成若干个文件来存储，那么下一个问题就是：怎么划分索引文件？

很容易想到利用作者名的首字母来划分索引文件，我们可以把索引文件划分为27个，对应着26个英文字母再加上一个其他字符，对于相同首字母（忽略大小写）的作者名，我们就可以把这些索引记录存放在同一个文件中。例如，假如有一个作者名叫做"Bert F. Lenting"，我们就可以把这个作者的索引记录放在b.csv文件中，这样我们在查询"Bert F. Lenting"这个作者时，就只用去b.csv中查找就行了。

对于存放标题记录的索引，我们也是同样的思路，创建27个专门用于存放标题记录的索引文件，把具有相同首字母的标题存放在同一个文件中。这样的设计，有利于在较小的范围内去查找有用的信息，由此便提高了查询速度。

### 有序性

也许你已经发现，光是高划分度，查找索引记录还是很费劲的，就算你的索引记录从原来的200万条缩减到了10万条，你还不是要逐个去遍历记录，检查是不是要找的目标记录，如果要找的记录是最后一个，那你要去遍历10万遍吗？因此，为了解决这个问题，我们将索引变成“有序”的，以符合二分搜索的条件。

二分搜索的应用，使得我们在大规模索引记录中搜索目标记录，只需要比对十五条左右的记录，大大缩减了查询时间。

### 不可重复性

不可重复性是索引的基本特性，即没有两条或多条索引记录是属于同一实体的。在该项目中，就是指作者名为"Bert F. Lenting"的索引记录最多只能存在一条，这样的好处就是提高索引系统的稳定性和可维护性。

* + 1. 原数据文件的格式

原数据文件（dblp.xml）是一个xml文件，一共有三级标签（可能出现四级标签，但是可以忽略）。标签的大致层级关系如下图。

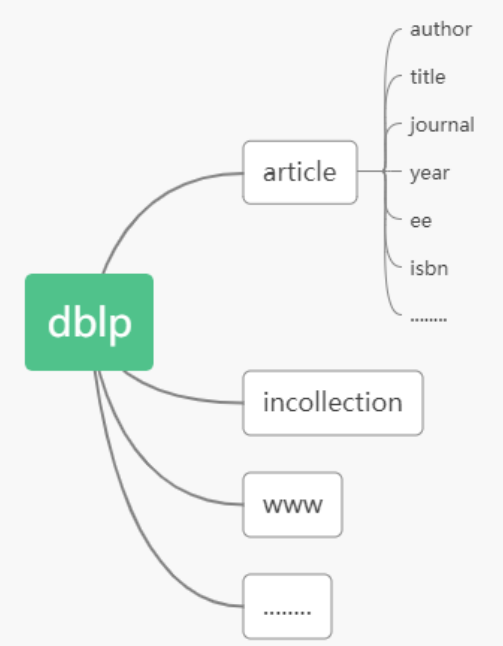


图1 dblp.xml 标签层级图

如上图所示，原数据文件只有一个一级标签<dblp></dblp>，二级标签中有article、incollection等，三级标签中值得关注的有title存放文章标题、author存放作者、year存放文章发表年份。了解了dblp.xml的大致结构，我们的索引创建过程可以大致分为三步：识别二级开始标签、提取二级标签内author、title等三级标签内信息、识别二级结束标签。

* 1. 初步索引文件的创建

初步索引文件只满足以上介绍的三大特性中的高划分度，即这时候的索引文件还不是有序的和不可重复的，下面简要介绍一下的初步索引文件的创建过程。

**伪代码1：初步索引文件的创建**

1. 初始化两个类型为Map<String, List<Long>>的authorIndex和titleIndex对象；
2. 读取原数据文件一段内容（内容长度大于任意二级标签内容长度）；
3. 寻找二级标签的开始标签，并记录这条记录的起始位置location；
4. 分别提取title和author三级标签内容authorName和titleName，以titleName为key、location为value存入titleIndex，以authorName为key、location为value，存入titleIndex；
5. 寻找二级标签的结束标签；
6. 检查authorIndex和titleIndex的大小是否大于阈值（该项目设为一百万），如果大于，则将authorIndex和titleIndex写入文件并清空authorIndex和titleIndex；
7. 检查是否到达原数据文件末尾，如果到达末尾，则退出程序，否则跳转到第二步。

**P.S. authorIndex和titleIndex的类型是HashMap<String, ArrayList<Long>>**

**为了提高存取的效率，index对象的类型是一个HashMap，它的key是作者名或者标题名，而且由于存在同个作者发表多个文章以及存在多个文章标题的现象，它的value必须是一个集合，这里取ArrayList<Long>。每当解析到一条记录，就将这条记录的(titleName, location)对存入titleIndex，一个或多个(authorName, location)对存入authorIndex中。**

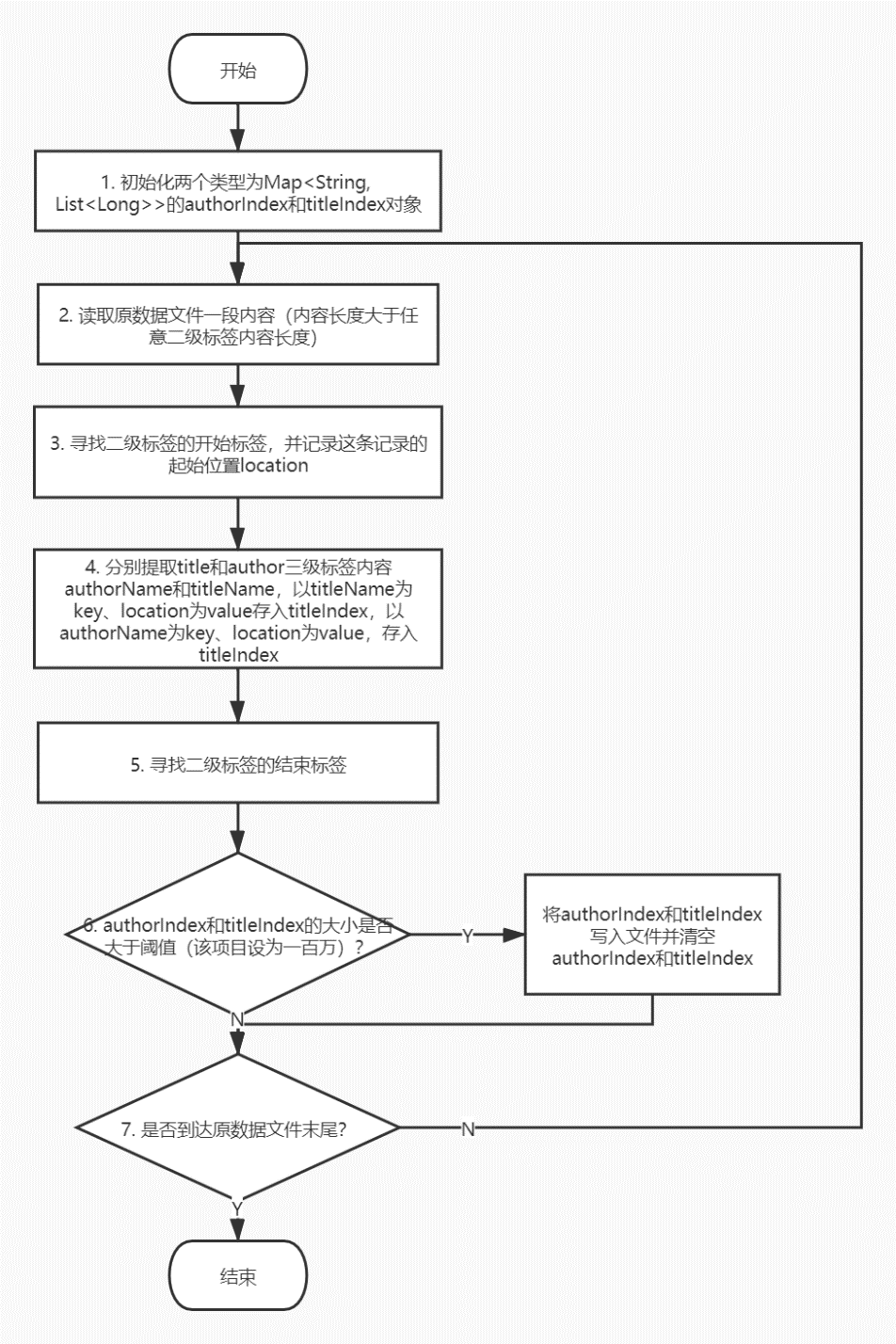


图2 初步索引文件的创建流程图

* 1. 最终索引文件的生成

如上所述，初步索引文件不具备有序性和不可重复性两大特点，因此我们必须要对此进一步加工，使其具备有序性和不可重复性。下面的伪代码展示了最终索引文件的生成步骤。

有序性的实现需要用到一种特殊的数据结构——红黑树，红黑树是一种平衡二叉树，具有搜索快、有序等特点，因此利用红黑树的特性，我们重新遍历索引文件，将无序的索引转化为有序的索引。

不可重复性也很容易实现，由于在遍历过程中，

**伪代码2：从单个索引文件中生成最终索引文件**

1. 初始化类型为TreeMap<String, List<Long>>的index对象；
2. 读取一行，提取出authorName或者titleName，以及locations集合；
3. 将(authorName, locations)对存入index对象中；
4. 检查是否到达原数据文件末尾，如果到达末尾，则退出程序，否则跳转到第二步。

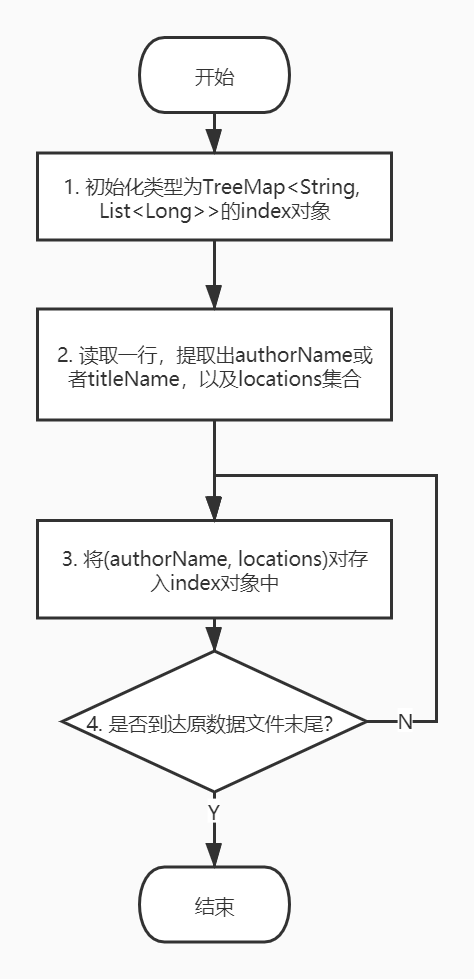


图3 最终索引文件的生成流程图

为了简要概述上述操作，请看下面一个例子，假设作者初步索引文件b.csv中含有如下两行：

"Berrin Dogusoy", 1033369700, 1337429294, 1595361903

"Berrin Dogusoy", 1599348286, 1601431793, 1855073934

注意到这两行的作者是同一个，而后面的索引集合locations是不一样的，那么经过从单个索引文件中生成最终索引文件后，生成的b.csv文件中就只存在与"Berrin Dogusoy"有关的下面一行：

"Berrin Dogusoy", 1033369700, 1337429294, 1595361903, 1599348286, 1601431793, 1855073934

对每一个索引文件都采取同样的操作后，得到的索引文件就同时具备了高划分度、有序性、不可重复性三大特性，为后续的算法设计打下了坚实的基础。

### 利用索引实现搜索功能

搜索功能是本项目一个基础的功能，看似简单，但是要想迅速地查找到目标记录，则对数据结构的设计要求非常高，多亏了索引文件，我们搜索功能的实现方式清晰又明了。本功能实现完全由本人设计及实现，测试结果表明确实能以毫秒级速度响应结果数据。

* + 1. 搜索功能实现

不管是根据搜索作者名搜索还是根据标题名搜索，都是四部曲：确定索引文件、在索引文件中通过二分查找法找到目标索引记录、提取索引记录的locations、根据locations在原数据文件（dblp.xml）中寻找响应的记录并做一些处理（搜索作者时可能要提取合作者等相关信息）。

在2.1.1中我们提到过，索引具有高划分度的特性，实现这一特性主要是利用了作者名或者标题名的首字母，相同首字母（忽略大小写）的作者名或者标题名的索引记录，都存放在同一个文件中。因此，我们便可以根据首字母去确定目标索引处于哪个索引文件。

以下展示了根据作者名查询作者发表的论文信息以及获取合作者的程序伪代码。

**伪代码3：根据作者名查找与该作者有关的信息**

1. 根据作者名的首字母确定目标索引位于的索引文件；
2. 在索引文件中通过二分查找法搜索到目标索引记录；
3. 提取索引记录的locations；
4. 逐个遍历locations，据此在原数据文件中查找相应记录；
5. 对查找的记录处理，提取出合作者等相关信息。

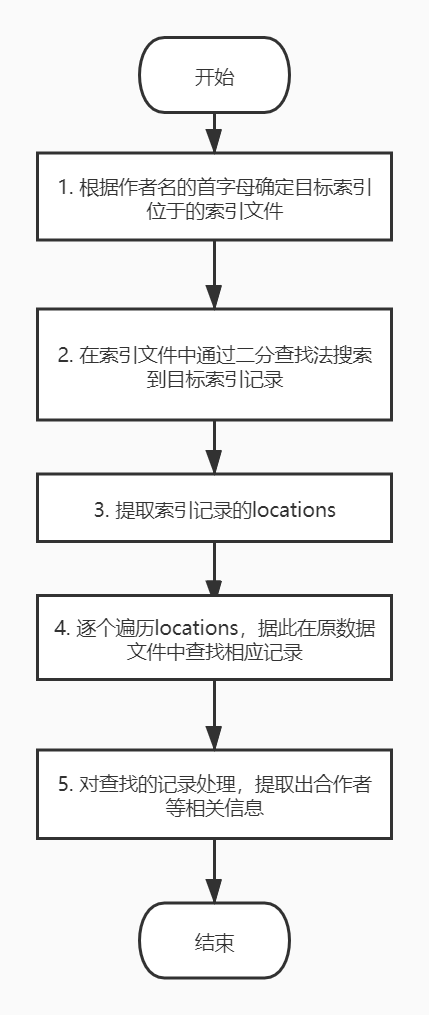


图4 根据作者名查找与该作者有关的信息流程图

上图是根据作者名查找与该作者有关的信息流程图，根据标题搜索相关记录的思路大致与此相同，故本报告不再赘述。

**P.S. 二分查找法的重要性**

**采用二分查找法，算法的时间复杂度就是**O(log(n))，显著降低了搜索时间，能实现毫秒级搜索效率。

* + 1. 搜索功能性能测试

为了展示搜索的极速，我们对根据标题搜索记录、根据作者名搜索记录及其合作者信息进行了测试。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试项 | 测试次数 | 总耗时/ millis | 平均耗时/ millis |
| 根据标题搜索记录 | 31 | 23 | 0.7419 |
| 根据作者名仅查找索引（不搜索记录） | 40 | 16 | 0.4 |
| 根据作者名搜索记录及其合作者信息 | 40 | 41 | 1.025 |

表3 搜索功能性能评测表

**可以看出，通过使用二分查找法和索引，基本的搜索算法表现出的性能十分强悍，可以在毫秒级时间内搜索到目标记录。根据标题名定位索引并根据索引查找记录平均耗时小于1毫秒，而根据作者名查找作者信息也才仅花费1毫秒左右。**

### 部分匹配功能

该功能由组员实现，在此引用他的部分报告内容。

* + 1. 使用正则表达式实现部分匹配功能（前期思路，先已废弃）

采用正则表达式进行部分匹配，同时在匹配的时候采用多线程进行速度的优化，考虑到如果是选择在磁盘文件里面进行遍历读取（也就是涉及到大规模的磁盘IO操作）会很慢，所以我们在首页的时候可以先点击按钮进行初始化，为后面的部分搜索提前做好准备，提前准备的部分先遍历title文件夹中的27个csv索引文件，接着初始化一个List<Article>类型的列表，其中的Article是自定义内容，负责记录文章标题、在源文件中的索引位置，不记录具体的文章内容（减少内存占用）。之后为了不让该列表被JVM垃圾回收，我们在项目的一个地方中用Scheduled注解配置了一个定时器在每一分钟都保持对其的引用，这样初始化一遍之后，只要项目保持tomcat服务器运行，就可以一直保持对其的引用和不被销毁。

其中正则表达式采用断言、懒惰匹配以及元字符\b的模式进行匹配。

|  |  |
| --- | --- |
| . | 匹配除“\r\n”之外的任何单个字符。要匹配包括“\r\n”在内的任何字符，请使用像“[\s\S]”的模式。 |
| \* | 匹配前面的子表达式任意次 |
| ? | 匹配前面的子表达式0到1次 |

表4 正则表达式匹配规则表

* + 1. 使用倒排索引实现部分匹配搜索功能
       1. 正向索引与倒排索引

在搜索引擎中每个文件都对应一个文件ID，文件内容被表示为一系列关键词的集合（实际上在搜索引擎索引库中，关键词也已经转换为关键词ID）。例如“文档1”经过分词，提取了20个关键词，每个关键词都会记录它在文档中的出现次数和出现位置。

得到正向索引的结构如下：

“文档1”的ID > 单词1：出现次数，出现位置列表；单词2：出现次数，出现位置列表；…………。

“文档2”的ID > 此文档出现的关键词列表。

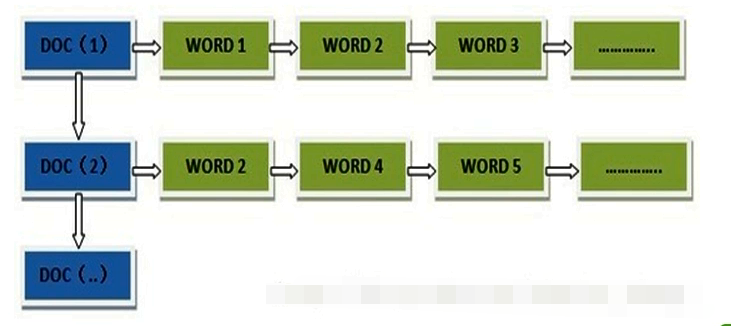


图5 正向索引结构图

当用户在主页上搜索关键词“华为手机”时，假设只存在正向索引（forward index），那么就需要扫描索引库中的所有文档，找出所有包含关键词“华为手机”的文档，再根据打分模型进行打分，排出名次后呈现给用户。因为互联网上收录在搜索引擎中的文档的数目是个天文数字，这样的索引结构根本无法满足实时返回排名结果的要求。

所以，搜索引擎会将正向索引重新构建为倒排索引，即把文件ID对应到关键词的映射转换为关键词到文件ID的映射，每个关键词都对应着一系列的文件，这些文件中都出现这个关键词。得到倒排索引的结构如下：

“关键词1”：“文档1”的ID，“文档2”的ID，…………。

“关键词2”：带有此关键词的文档ID列表。



图6 倒排索引结构图

* + - 1. 倒排索引法实现思路



图7 倒排索引文件结构图

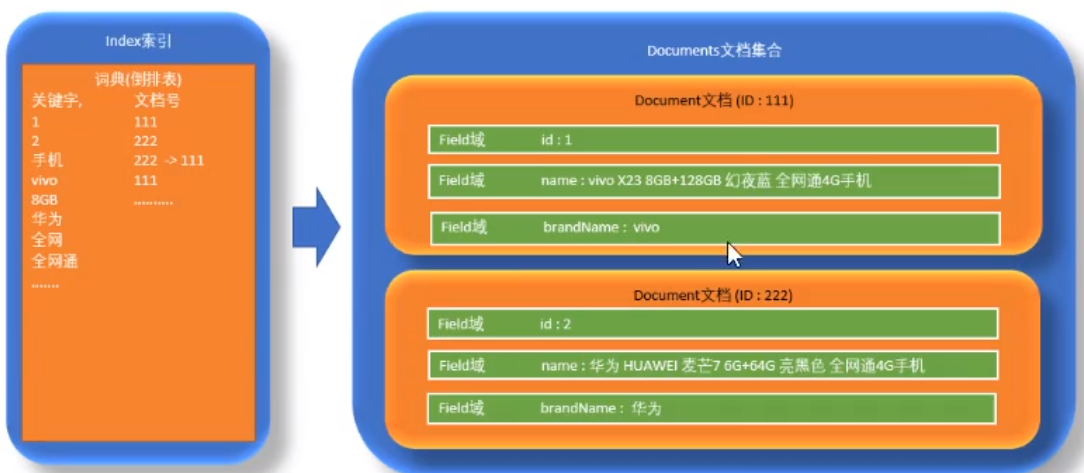
我们先对文档建立倒排索引，上图是倒排索引文件结构图。索引提取了资源中关键信息，并以文件形式存储在磁盘里。使用部分匹配搜索标题记录时，根据关键字（目录），快速定位索引，并根据索引查找在原数据文件中的记录。以下是部分匹配功能的算法描述。

查询前会先将查询的内容提取出来组成文档（正文）,对文档进行切分词组成索引（目录）,索引和文档有关联关系，查询的时候先查询索引，通过索引找文档的这个过程叫做全文检索。

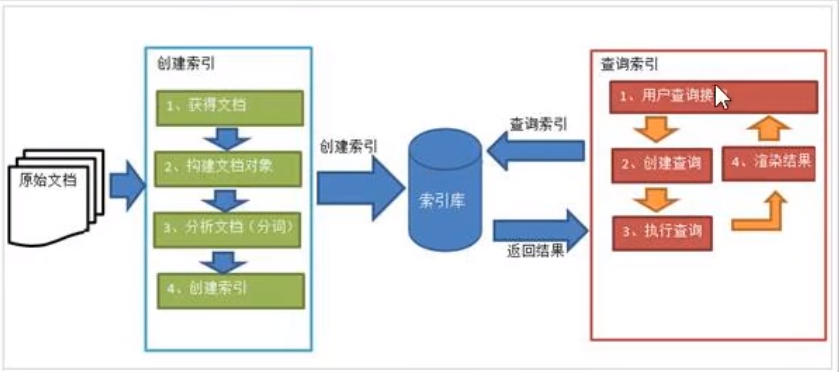
切分词：就是将一句一句话切分成一个一个的词，去掉停用词（的，地，a，an，the等）。去掉空格，去掉标点符号，大写字母转成小写字母，去掉重复的词。

倒排索引相比顺序扫描的优点

因为索引可以去掉重复的词，常用的英文单词在牛津词典也有收录。如果用计算机的速度查询， 牛津词典这些内容是非常快的.但是用这些字典，词典组成的文章却是千千万万不计其数.索引的大小最多也就是牛津词典.所以通过查询索引，再通过索引和文档的关联关系找到文档速度比较快。而顺序扫描法则是直接去逐个查询那些不计其数的文章，所以遍历的速度会很慢。

图8 倒排索引的存储结构图

在搜素的时候，我们会对查询的关键词数组数组进行切分词，再根据切分出来的关键词数组到Index索引文件中找到文档ID列表，再根据列表到Document文档集合里面找到符合查询条件的title。

图9 索引和搜索的过程图

### 作者统计功能

该部分功能实现完全由本人实现。统计发表文章数最多的前100位作者其实就是一个简单的排序问题，只要按作者的发布文章数排序，就能轻松解决这个问题。该项目使用了桶排序算法，来实现O(n)的时间复杂度。

**伪代码5：统计发表文章数最多的前100位作者**

1. 获取所有作者的发表文章数；
2. 初始化足够数量的桶buckets；
3. 将每个作者按照发表文章数articlesNumber存放到bucket[articlesNumber]上；
4. 按下标从大到小的顺序遍历buckets，提取发表文章数最多的前100位作者。

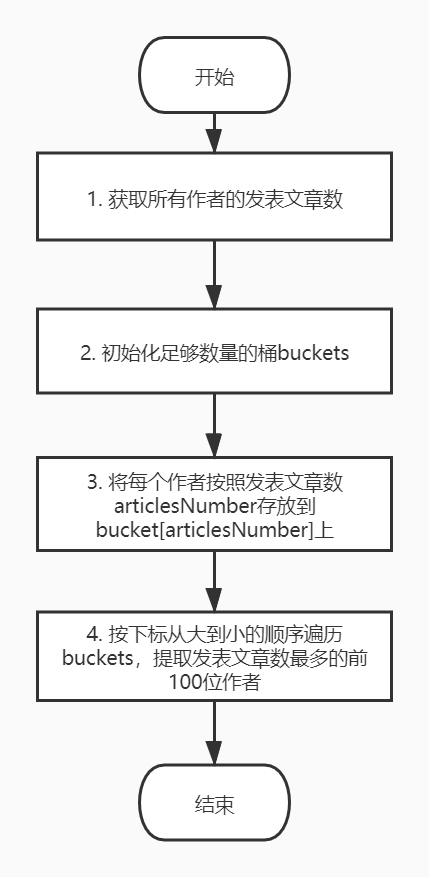


图10 统计发表文章数最多的前100位作者流程图

### 实现热点分析功能

该功能由组员实现，在此引用他的部分报告内容。

### 生成热词排行榜

我们首先通过前面所使用的读取文件方式，将文献标题和文献发表年份形成一个Hashtable储存起来。即构造一个Hashtable<Interger,ArrayList<String>>类型来储存数据。结构如下图所示：

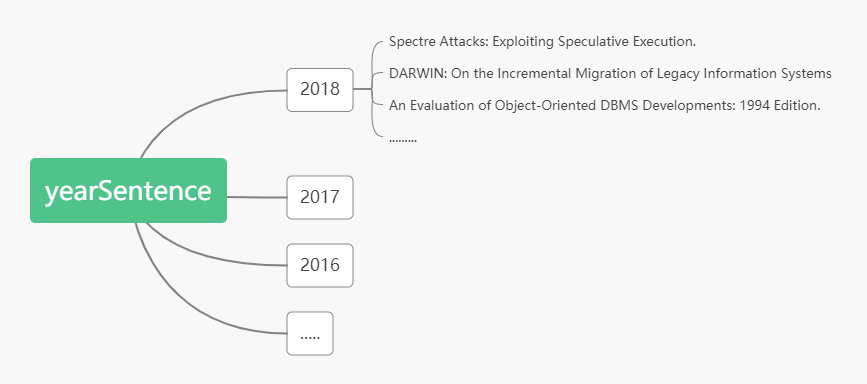


图11 热词分析处理数据结构图

提取出了yearSentence之后，我们所要做的就只是针对于每一年的所有词汇，提取出当年的年度最热词。但是这里还存在一个尚未解决的问题：句子如何划分成为单词？

这里存在很多种解决方法，我们采用的是使用较为简便的StringTokenizer来将字符串分解，同时对于每年的所有title，将分解出的单词计次加入到类型为<String, Integer>的Hashtable中。之后通过遍历一次此前的Hashtable来确定此年度最热门的前十个单词，并将其和这一年的年份储存到<Integer,Hashtable<String,Integer>>类型的hashtable中。

但是这样提取出来的年度最热词有一个不可忽视的问题：其中很多词汇是无意义的语气词，例如：“for”，“of”，“on”等等。针对这个问题，思路如下：

1. 调用nlp有关的库和函数，判断分割出来的单词词性，排除介词词性的单词。

但是在实施的时候我们发现，出现频率高的无意义词，可能并不只是介词，其还有许多其他词性。而且部分无意义词的排除可能导致该词性的部分有意义词被舍弃。并且在判断词性的过程中，由于需要调用包和函数，代码的效率降低。

1. 选取出现频率高的无意义词，并在计次时取消它的计次机会。

对于此时我们得到的结果进行分析，我们可以发现，其中占用热词的，多为少数几个词。因此我们人工选取了结果中出现的无意义单词，并将其屏蔽。最后得出的结果效果还不错，而且代码的效率较高。因此我们选用了该方法。

### 热点词汇的持久化存储

由于我们所拥有的数据文件是固定且不会发生改变的，所以每年的热点词汇也是固定的，没有必要每次从头开始分析数据进行读取。所以我们采用了一个txt文件格式来存取 点词汇，方便可视化的时候进行读取。

### 聚团分析

该功能由组员实现，在此引用他的部分报告内容。

### 聚团分析解决思路

聚团分析是一个相对来时比较难以实现的功能，因为要统计整个图的各阶完全子图的数量，而这个图有250万左右的节点数量。统计一个节点数量极大的复杂网络图的各阶完全子图的数量，对算法的时间效率和空间利用率是一个极大的挑战。

算法步骤：

1.构建基于链表的无向图邻接表

2.while循环遍历每个节点（找出包含此节点所有不重复的最大完全子图）：

For循环从这个节点的不同邻居节点遍历：

选定一个邻居节点，for开始对剩下的节点遍历：

若这个邻居顶点是否与先前的完全子图构成更大的完全子图，是则加入该完全子图

对获取的完全子图，排序（为了更好的去重），判断该完全子图是否已经从其他节点纳入统计，若否则加入结果链表。

3.统计完成，释放资源

### 代码结构

根据选定的算法，我将代码实现分为以下几块，以下是聚团分析代码之间的而关系图：

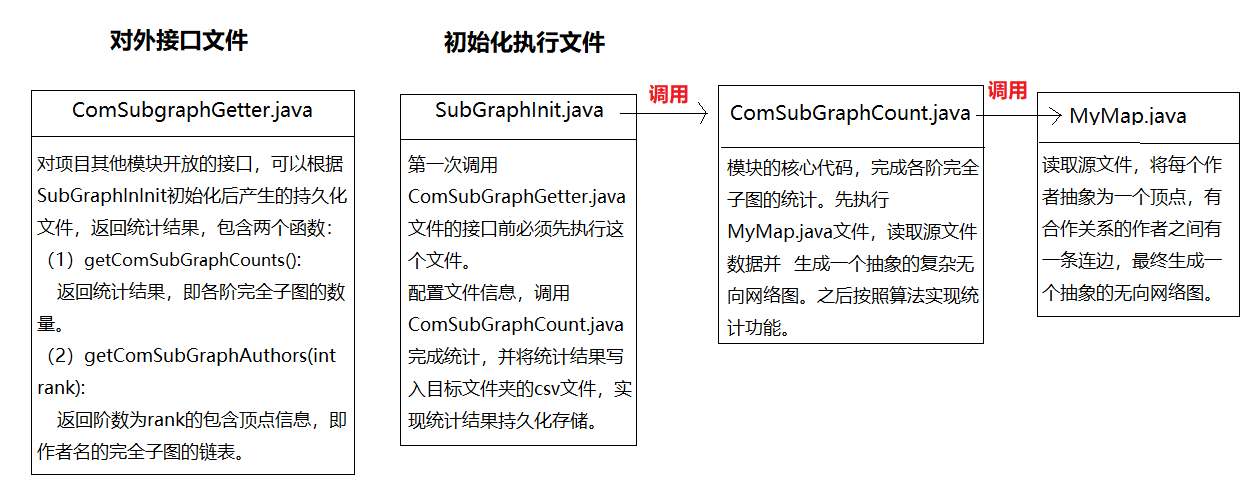


图12 聚团分析代码结构图

### 聚团分析代码实现

（1）构建作者合作关系无向图

1. 用随机读取方式打开源文件，新建一个myMap记录图的信息；
2. While循环，当还未读到文件末尾时：

找到一个文献记录，对其中的每个作者分配一个自增长的顶点id，并将该id与该作者名的映射追加到nameMap。在myMap中，对同一个文献的作者顶点之间，两两互连形成一条边。

1. 将nameMap写入指定位置的csv文件，并返回myMap；

（2）生成各阶完全子图

1. 获取源文件路径并调用MyMap.java生成抽象无向图；
2. 对无向图中的每个顶点：
   1. 如果该节点度数为0，则纳入一阶子图
   2. 若度数>0, 找出包含此节点的所有完全子图，并纳入统计:

分别从此节点的不同邻居节点遍历：

选定一个邻居节点，开始对剩下的邻居节点遍历：

若这个邻居顶点是否与先前的完全子图构成更大的完全子图，是则将该节点加入该完全子图

对获取到的完全子图排序（为了更好的去重），判断该完全子图是否已经从其他节点纳入统计，若否则加入结果链表。

1. 释放资源返回结果。

### 可视化技术

该项目的页面展示功能主要由本人实现。该项目使用SpringMVC技术进行可视化界面展示，采用B-S架构，任何电脑都可以通过访问网页向服务器获取数据。SpringMVC提供的简易性使得该项目的开发变得十分简单，只需要前端页面发送ajax请求，请求就会被服务器接收并处理，最终返回所需要的数据，将数据在前端页面渲染后就可以进行展示。在结果分析部分将有本项目的所有页面展示。

由于本报告主要聚焦于实现该项目所用的数据结构和算法设计，故本部分内容不赘述。

若想访问项目网址，请联系本人启动服务器。

### 结果分析

### 文章列表页面展示



图13 文章列表页面展示图

上图展示了该项目的文章列表页面，页面下方的表单中每一行展示了文章标题以及该记录在xml文件中的索引，点击右边的“查看详情”按钮可以查看详细的记录信息。



图14 查看文章详情记录展示图

在页面上方的输入框中输入准确的文章标题全名（不可包含特殊字符，如法文）就可以搜索到对应的记录。



图15 根据文章标题搜索记录结果展示图

### 作者列表页面展示

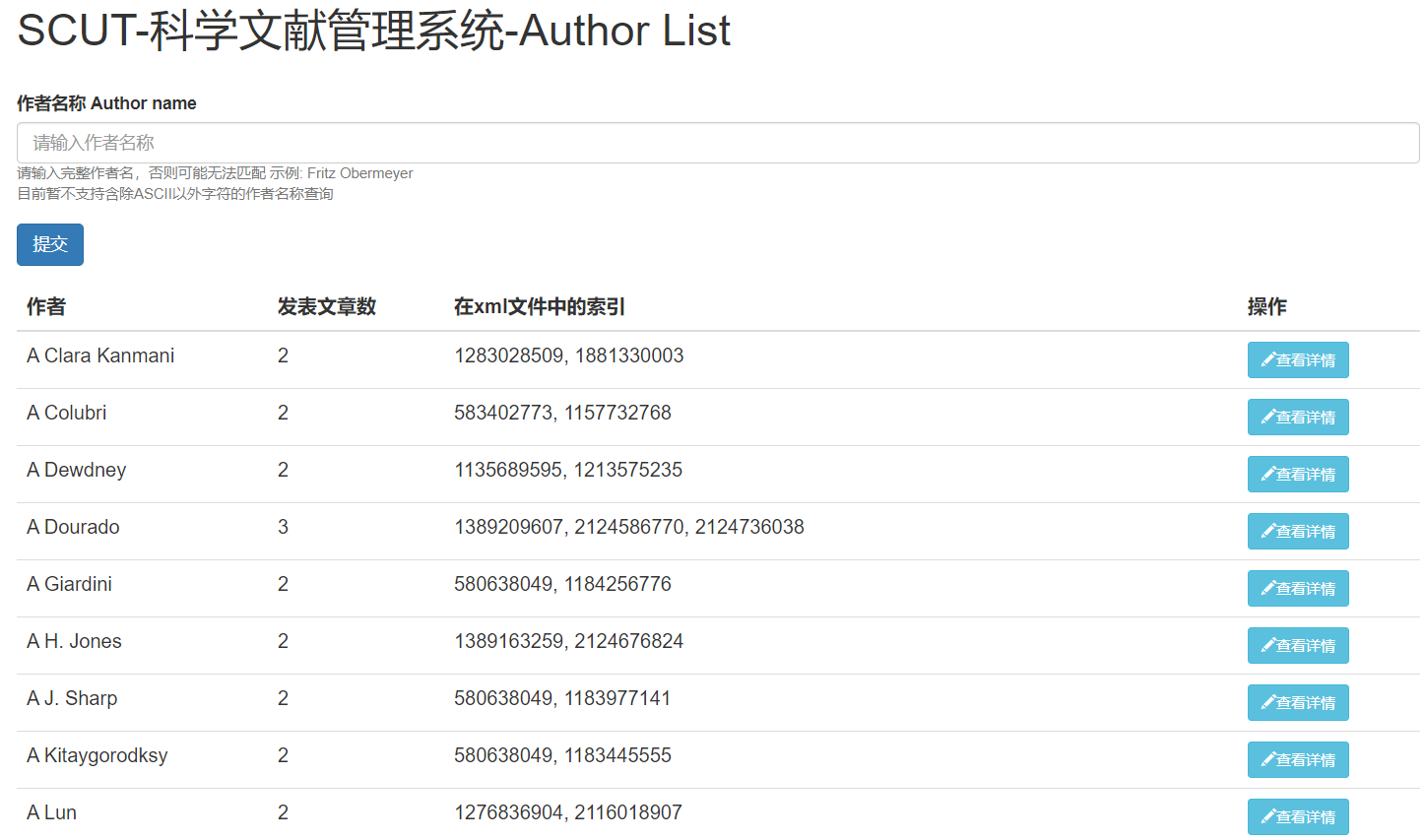


图16 作者列表页面展示图

作者列表页面与文章列表页面大同小异，唯一不同的是表单增加了“发表文章数”这一列，点击“查看详情”按钮可以查看到该作者发表文章的详细记录信息以及他的合作者。



图17 查看作者详情记录展示图

### 作者合作关系图展示

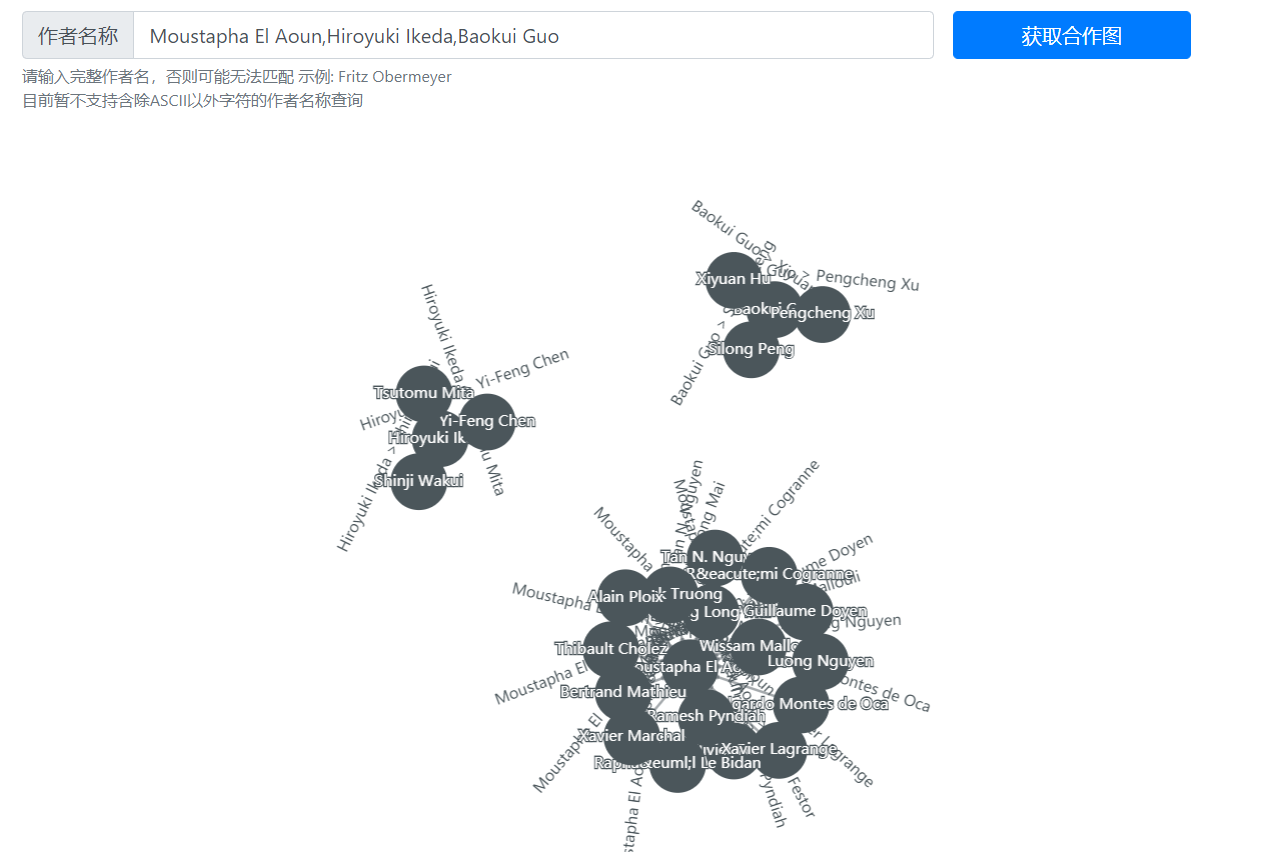


图18 作者合作关系页面展示图

作者合作关系图页面很简洁，在上方输入框中输入一个或多个作者名称就可以显示作者关系合作图。

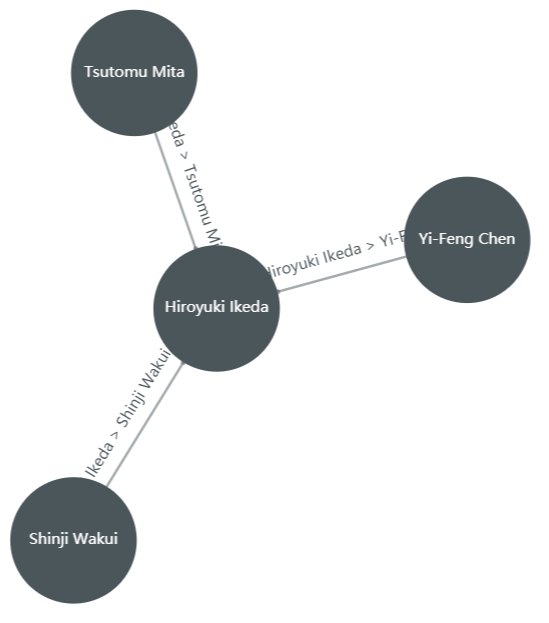


图19 作者关系合作图

仔细查看作者合作图，可以看出查找的作者位于图的中央，边表示起点作者与终点作者有合作关系。

### 发表文章数作者排行榜页面

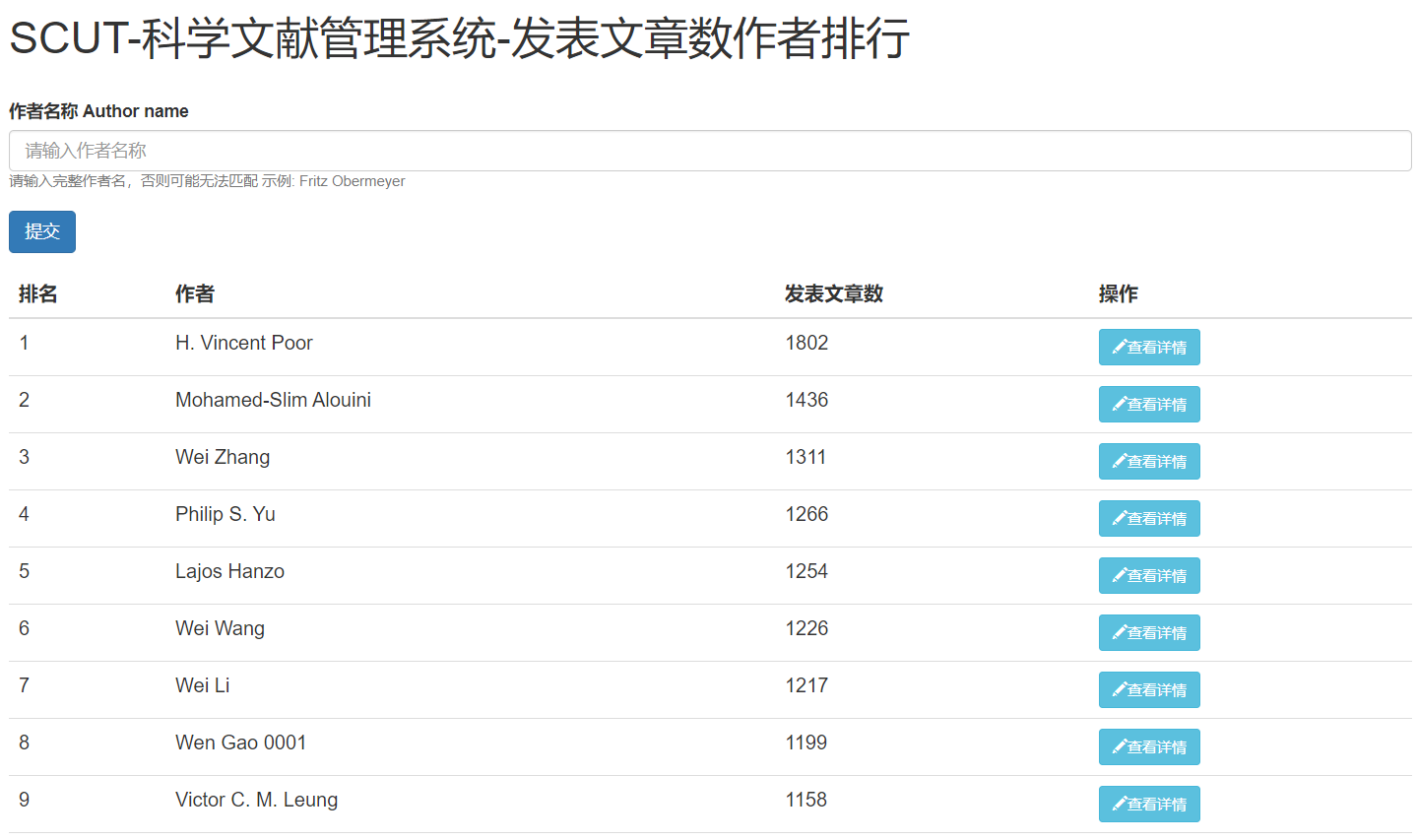


图20 发表文章数作者排行榜页面展示图

### 年度热词页面展示



图21 年度热词页面展示图

从首页进入年度热词页面，点击“生成年度热词表格”按钮，就可以看到如上页面，表单中的每一行展示了该年份的top10热词以及对应的出现频次。

### 聚团分析页面展示

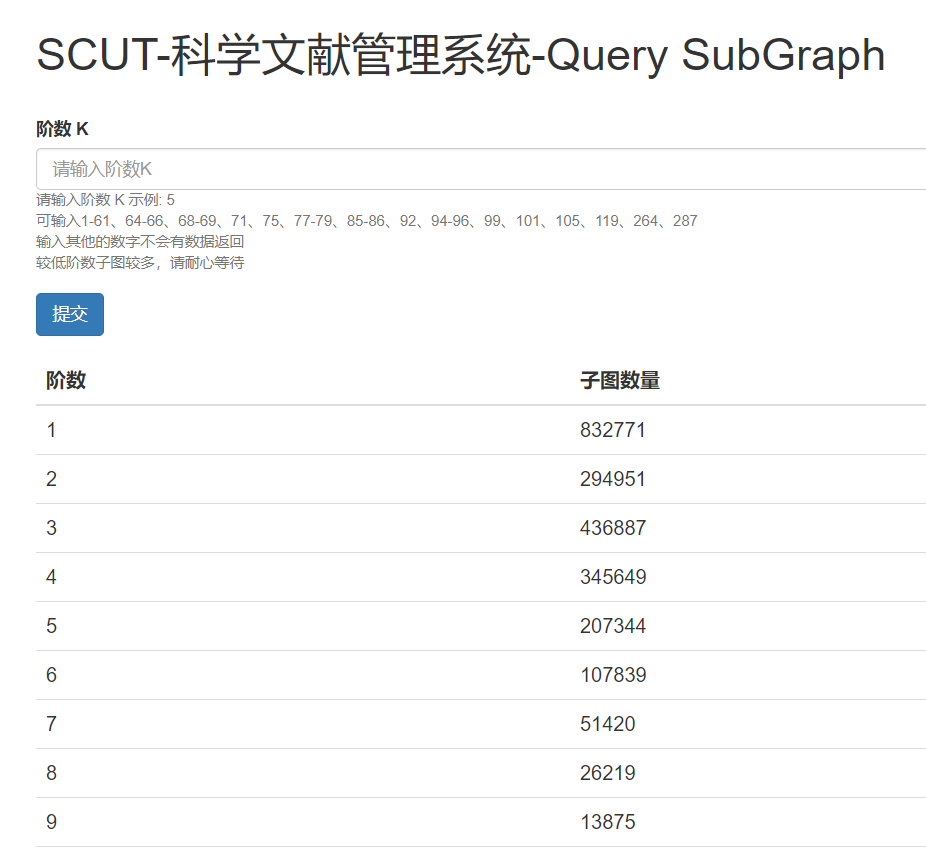


图22 聚团分析页面展示图

聚团页面展示如上图所示，表单显示了各阶子图的数量，在上方搜索栏中输入阶数，就可以查询该阶数的所有完全子图。如下图展示了阶数为30的所有完全子图，“完全子图”这一栏表示该完全子图中所包含的作者，作者数量等于阶数。

、

图23 查看k阶子图功能展示图

### 课程设计总结（格式：宋体，4号，加粗，两端对齐）

总结可以包括:课程设计过程的收获、遇到的问题，遇到问题解决问题过程的思考、程序调试能力的思考，课程设计实现过程中的收获和体会等。

（正文格式：宋体，小4号，不加粗，两端对齐，行距为固定值20磅）

### 善用索引

我在小组内的工作包括索引文件的创建，所以我非常熟悉该项目中索引文件的工作原理以及背后的奥秘。在三.2. 索引创建方式详解中我提到索引的三大特性：高划分度、有序性、不可重复性，其中高划分度缩小的索引的搜索空间，有序性保证了二分查找法的有效性，不可重复性保证了搜索算法的准确性。这三大特性共同造就了该项目搜索的极速性能，是该项目的“基础设施建设”，一个优秀的索引创建，为后续的开发工作打下了坚实的基础。

那么，我是怎么想到用索引来完成搜索功能的呢？其实，本项目一开始是没有索引文件的，要想进行搜索就必须先扫描原数据文件（dblp.xml），把记录全部以哈希表的形式读取到内存中，其结果可想而知，读取高达2G的文件就要花费不短的时间，而把索引维护在内存中，更是消耗不小的内存，因此本方案很快就被废弃了。随后我又根据记录的特点，以作者名或者标题名为key，记录地址为value，创建了一组索引文件。其运行效率着实令人震惊，能在毫秒级时间内完成所需操作。

因此，我在本次课设学到的一大重要知识就是善用索引，通过索引，便可以快速定位目标记录在原数据文件中的位置，从而实现极速搜索。

### 二分查找法的妙用

**光有索引还不够，索引只是可以通过索引中记录的地址去查找详情记录，但是如何快速查找到索引呢？如果采用时间复杂度为O(n)的遍历方式，在10万条索引记录中查找目标索引，就会导致性能的瓶颈，使得搜索速度十分之慢（约1秒）。于是，我设计出了一种对索引文件的二分查找法，只需要比对不超过20条记录，就可以找到目标索引，可以说是性能的又一大进步。通过索引+二分查找法，便真正地实现了极速搜索、快速定位的功效。**

### 聚团分析如何实现

聚团分析的设计实现运用到了图的相关算法，通过初始化一个邻接表，表示每个作者与其他作者的合作关系，随后在这个邻接表中进行完全子图的扩增处理，便可以较为简单的实现聚团分析功能。

### 部分匹配搜索功能带来的启示

通过所给的关键词进行部分匹配搜索，得到相关的文章记录，这个功能看似简单，其实不容易实现。我们小组的做法首先是在内存中维护一个索引哈希表，存储文章标题和相应的地址，实际上这种做法不太科学，因为要想使用部分匹配搜索功能，就先要去初始化索引哈希表，对CPU和内存的消耗过大，因此这种方法后来就被废弃掉了。最终一个比较好的实现方案是使用一个倒排索引（见三.6.2. 使用倒排索引实现部分匹配搜索功能），测试结果表明此方案是成功的，能够较快速的搜索到相关的文章记录。

# 参考文献

1. 张洁. Java解析xml文件的研究与应用[J]. 硅谷, 2014, 000(006):120-120,128.塞奇威克, R.), 韦恩,等. 算法: 第4版 : 英文[M]. 人民邮电出版社, 2016.
2. 严蔚敏，吴伟民. 数据结构(C语言版)（第二版）. 人民邮电出版社.2016.
3. 殷人昆等. 数据结构（用面向对象方法与C++语言描述）第二版. 清华大学出版社. 2007.
4. 韩森, 邓停东, 张占强. SpringMVC＋Hibernate框架的应用研究[J]. 教育技术导刊, 2009, 000(006):44-45.
5. CliffordA.Shaffer, Shaffer, 张铭,等. 数据结构与算法分析[M]. 电子工业出版社, 2010.