# 校园搜索引擎构建

马　也　2013011365 计 34
刘政宁　2013011362 计 34

June 17, 2016

# 1 实验要求和内容

本项目要求综合运用搜索引擎体系结构和核心算法方面的知识，基于开源资源搭建校园搜索引擎，掌握开源搜索引擎的运行流程。具体要求为：

1. 抓取清华校内绝大部分（30 万左右）网页资源及大部分在线文本资源（如 office 文档、pdf 文档等）
2. 实现基于 BM25 概率模型的内容排序算法，要求对查询进行分词；
3. 实现基于 HTML 结构的分域权重计算（content/title/h1-h6），并应用到搜索结果排序之中，并建立小规模测试集合，进行参数调节；
4. 实现基于 Page Rank 的连接结构分析功能，离线计算 Page Rank 值，并应用到搜索结果排序之中；
5. 采用便于用户信息交互的 Web 界面，实现查询扩展、查询纠错等功能。

# 2 实验功能

本项目在全部完成基础要求的同时，最终完成如下扩展功能，具体功能描述及效果参见实验成果一节。

- 基于 HTML 结构的分域权重计算（包含 content/title/h1-h6 等），不同域权值不同
- 实现了条件查询功能，即支持不同查询语句的 AND、OR、NOT 组合
- 实现了模糊查询功能，即支持自动纠正错误输入，并得到正确查询结果
- 实现了通配符查询（正则表达式）功能，即支持 * ？ 等~ 通配符进行查询
- 实现了查询特定范围功能，即支持查询某一网域下的相关网页
- 实现了查询特定类型功能，即支持查询某一或某几类型的资源
- 实现了查询结果界面的高亮处理，即找到最佳高亮位置，显示到摘要之中
- 实现了对特定 HTML 结构的查询，即支持仅在标题、h1 等结构中查询

# 3 实验框架及运行环境

## 3.1 框架概要

本项目主要分为五个模块：爬取模块、预处理模块、索引模块、查询解析模块以及 Web 模块。主要流程关系参见图3.1，具体阐述如下：

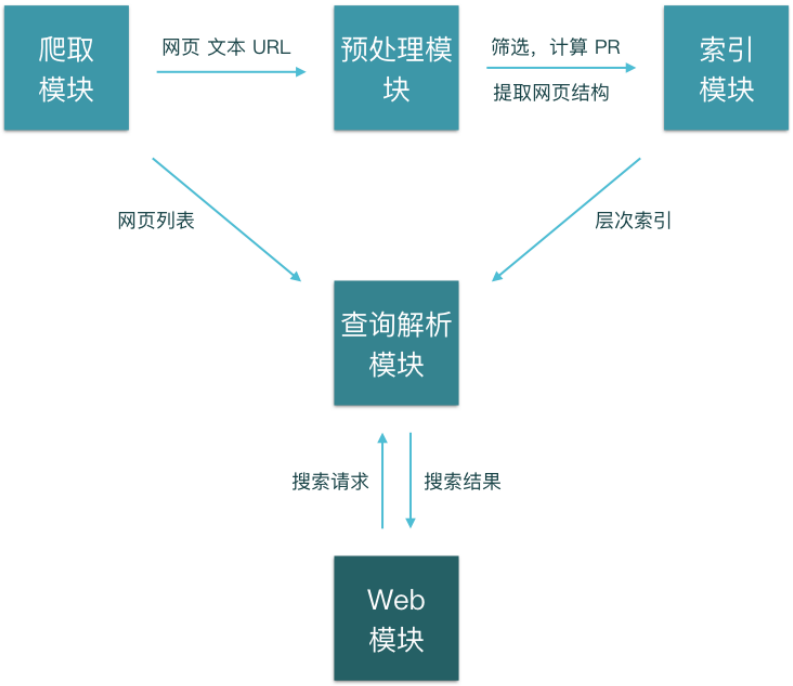


Figure 3.1: 实验框架

**爬取模块**从清华校内爬取网页和文本数据，并按照网页结构（url）分文件夹存放在本地，对应搜索引擎中的数据抓取子系统；

**预处理模块**分析和处理爬虫爬取到的数据，筛选高质量网页，提取网页 HTMl 结构到纯文本文件之中，交由索引模块使用，并分析网页链接结构，计算 Page Rank 值，其对应搜索引擎中的链接分析子系统和内容索引子系统的一部分；

**索引模块**则利用预处理模块提取好的文本文件，使用分词器构建多层次索引，存储各层结构的文本信息到索引数据库中，以备查询模块使用，其对应搜索引擎中的内容索引子系统；

**查询解析**模块利用建立好的索引，根据查询条件和要求构建相应的查询语句，分拆高级查询语句为基本查询语句的组合，并在索引数据库中进行查询，返回查询到的文档列表，其对应内容检索子系统；

**Web 模块**代表了搜索引擎所对应的网站，其负责接收用户请求，发送给查询解析模块，收到结果后以合理的格式与结构返回给用户。

## 3.2 爬取模块

爬取模块使用第三方库 Heritrix 3.2.0 完成，Heritrix 可以对抓取的对象进行精确的控制，很好地符合我们校园搜索的要求。需要指出的是，最早项目使用课件上使用的版本，但其爬取速度太慢，且正则表达式过滤模块有隐含的 BUG，所以最终替换为更新的版本，因此配置和课件上的配置有所不同。主要配置了如下内容：

- 种子资源：http://news.tsinghua.edu.cn/
- 接受网页总规则：以 tsinghua.edu.cn 结尾，且不以 lib.tsinghua.edu.cn 结尾，不属于 166.111.120 网段
- 拒绝类型规则：js|JS|axd|AXD|mso|tar|txt|asx|asf|bz2|mpe?g|MPE?G|tiff?|gif 等，从略

除此之外，还设置了爬取速度，最大条数等具体配置，最终爬取了 40 万以上的文件，删除掉其中大于 32M 的较大文件，最终剩余 35 万左右。

### 3.3 预处理模块

预处理模块基本使用 Python 3 完成，计算 Page Rank 使用 C++ 完成，代码在 preprocess 文件夹内，脚本文件及对应功能如下：

- **parse_log.py**：分析 Heritrix 日志，建立文件名到网页 URL 的双向映射关系
- **get_id.py**：分析链接结构，筛选有效网页，给每个网页分配独立 id，并得到链接图谱
- **append_id.py**：对 get_id 的补充
- **prepare_pr.py**：将 Python 结构按照规则写入文件，为 C++ 程序准备输入
- **page_rank.cc**：计算 Page Rank，写入文件中，每行一个 id 和对应的 Page Rank
- **get_text.py**：提取网页文本内容，删除 script、css 以及 HTML 标记，每个网页存入一个文本文件中，文件名为网页 id
- **get_title.py**：提取网页标题（<title> 域），写入文件中，每行一个 id 和对应的 title
- **get_file_text.py**：提取 PDF、DOC、DOCX 等格式的全部文本，每个文件存入一个文本文件中
- **get_file_title.py**：提取 PDF 文件的标题，写入文件中，每行一个 id 和对应的 title
- **get_docs_title.py**：提取 WORD 文件的标题，写入文件中，每行一个 id 和对应的 title
- **get_h1.py**：提取网页结构，（h1-h6），分别写入文件中

需要注意的是，由于 Heritrix 在抓取带 GET 请求的网页时，存储文件的文件名和网址 URL 并不能一一对应（其去掉了问号，挪动了文件类型的位置），且单从文件名并不能找到对应的 URL，所以第一步分析 Heritrix 爬取日志是必要且是必须的。通过分析日志，得到了 URL 到文件名的双向映射，同时删除了 404 网页，将网页个数减少到 32 万左右。

以上处理中对于 HTML 网页的处理使用 Beautiful Soup 完成，其负责提取链接关系，提取文本信息，提取 title 和提取 h1-h6 域等，部分编码混乱的网页被直接丢弃，将网页个数再减到 30 万左右。

另外，由于绝大多数下载附件网页的文件名都是数字编号或无意义符号，所以提取 PDF 和 DOC 文件的标题也是十分重要的。提取 PDF 内容使用了 pdftotext 程序，提取 PDF 标题使用了 pyPdf 库，通过 PDF 文件的 Meta 信息能够得到大部分 PDF 的准确文件名，提取 DOC 文件使用了 antiword 程序，提取 DOCX 文件使用了 docx 库。

## 3.4 索引模块

索引模块使用 Lucene 5.5 完成（MyIndexer.java），主要存储了网页 id、网页标题、网页内容（content/h1-h6 等）、网页类型（HTML/WORD/PDF 等）、网页 Page Rank 值到索引之中，其中网页标题、内容进行了分词处理，分词时使用了效果更好的 jcseg 第三方库，可以有效地区分数字、人名、专有名词等，分词准确率更高。

索引时使用的评分模块为 BM25 模型的改版（MySimilarity.java），BM25 模型在 Lucene 5.5 官方提供的 BM25Similarity 的基础上进行了重构，加入了 Page Rank 的计算，修改了最终得分公式，将 BM25 的得分与 Page Rank 的 0.5 次方乘起来得到最终得分，最终评分公式为：

$$\text{Score} = \text{idf} \cdot \frac{(k+1) \cdot \text{freq}}{k \cdot (1 - b + b \cdot \frac{|d|}{\text{avgLen}}) + \text{freq}} \cdot \sqrt{\text{PageRank}}$$

这里使用根号的原因是减少 Page Rank 对于结果的影响，因为 Page Rank 范围变化较大（$10^{-3}$ 到 $10^{-7}$），如果直接将 BM25 结果和 Page Rank 结果相乘，会导致 Page Rank 高的网页，即使只出现一次也会排名非常靠前，这就让 BM25 算法失去了意义。

## 3.5 查询解析模块

查询解析模块使用 Lucene 5.5 完成（MySearcher.java），主要实现了普通搜索（search()）、获得文本高亮（getHighlight()）、高级搜索（searchComplex()）、通配符和模糊搜索（searchFuzzyOrWildCard()）等功能。

需要注意的是，如果直接使用 QueryParser，其生成的 Query 语句的分词间是 SHOULD 的关系，即只要有一个出现即可，这不太符合大多数搜索的要求，如搜索“清华大学计算机系”时可能出现只含“清华大学”而不含“计算机系”的文档，所以要使用 QueryBuilder 类的 createMinShouldMatch 方法构建 Query，这样保证分词后的每个词都必须出现，而不是出现一个即可。

得到了基础的 Query 之后，需要根据要求再不同域上进行搜索（content、title、h1、type 等），并且需要使用 BoosQuery 提供不同域的不同权值。如果进行条件搜索，则需要 BooleanQuery 对其进行组合，BooleanQuery 中的 SHOULD、MUST、MUST_NOT 分别对应 OR、AND 和 NOT。

对于通配符搜索和模糊搜索，则可以使用 WildCardQuery 和 FuzzyQuery，按照类似的要求进行组合，得到最终的 Query 查询。

文本高亮也是在这一模块实现的，主要使用 Lucene 中的 Highlighter 类，对于一个特定的 Query、特定的 field 和和内容，可以找到最佳的文本段，其出现特定关键词的次数最多、得分

最高，将其返回给前端，就可以实现类似于 Baidu、Google 等搜索引擎的文本高亮了。

### 3.6 Web 模块

该模块使用 Tomcat 服务器完成（IndexServlet.java 和 ResultServlet.java），主要实现了 myIndex.jsp 和 myResult.jsp 两个动态页面，其负责显示搜索结果，进行分页处理等操作，并接受用户高级搜索的输入，转化为查询解析模块的函数调用。

## 4 实验成果与分析

### 4.1 主要功能与效果图

#### 4.1.1 主界面

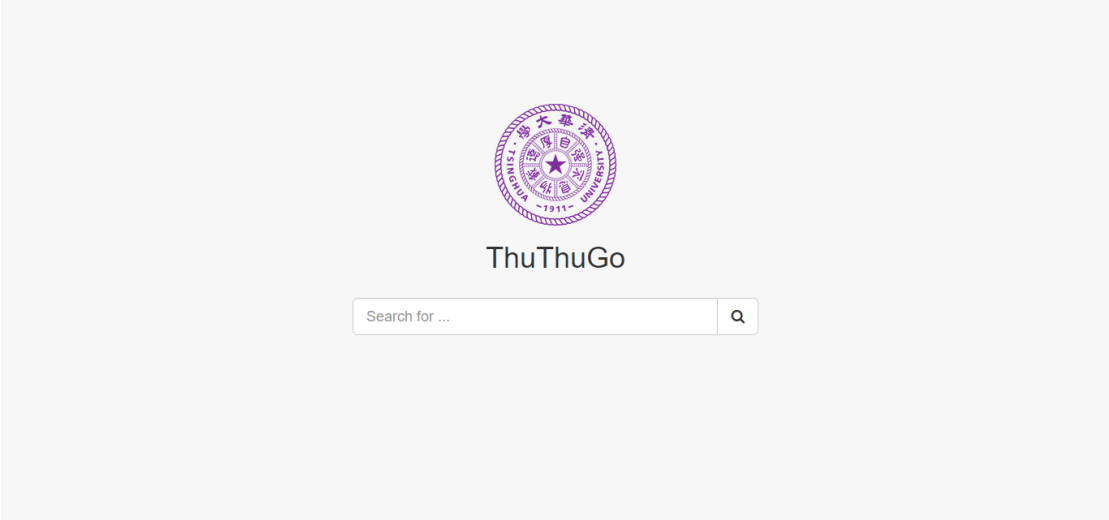


Figure 4.1: 主界面

#### 4.1.2 搜索结果

#### 4.1.3 搜索文件功能

#### 4.1.4 模糊搜索功能

#### 4.1.5 通配符搜索功能

#### 4.1.6 条件搜索功能

### 4.2 实验结果分析

#### 4.2.1 入链接、出链接分布

首先，对抓取到的网页进行入链接、出链接统计，可以得到它们的分布情况，如下图所示：

从中我们可以看出，对于入链接个数分布情况来说，较好的满足了幂律，即入度与出现次数满足指数函数关系；而对于出链接个数分布来看，其在低频区不能很好的满足幂率，但是依然有着相似的分布。

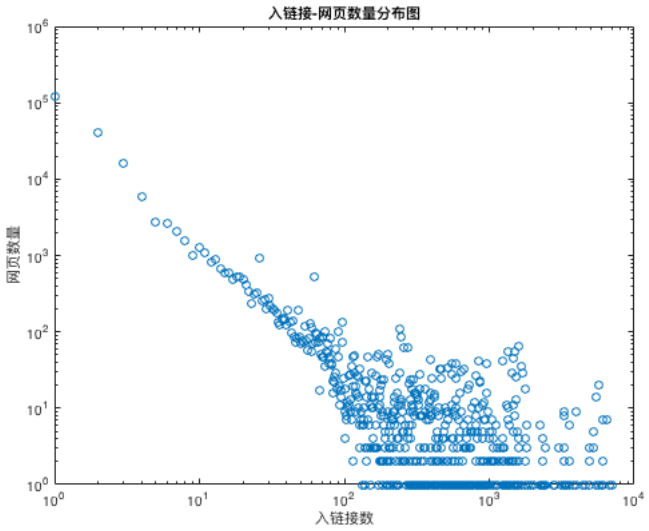


Figure 4.2: 入链接分布情况

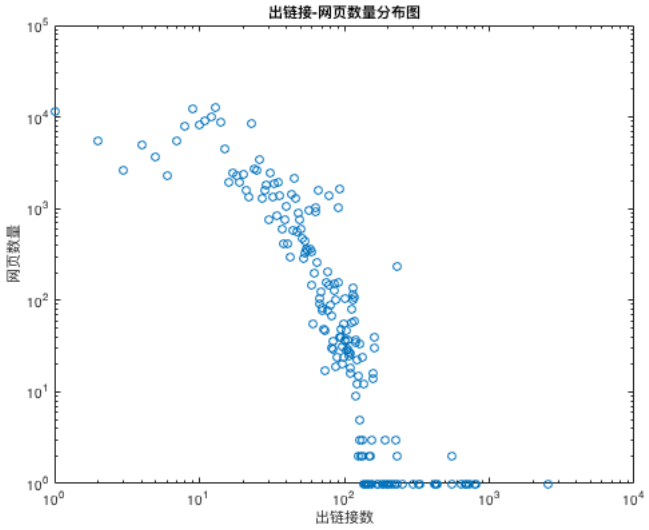


Figure 4.3: 出链接分布情况

## 4.3 PageRank 算法结果

使用编写的 compute_page_rank.py 程序，我们可以得到每个节点的 Page Rank 值。在计算中，我们取 $\alpha = 0.15$，$TN = 30$，即经过 30 轮迭代之后，得到每个节点的 Page Rank 值，并将其和该节点的入度、出度信息一同写入文件 pr.out 之中。再使用 sort_page_rank.py 文件将节点按照 Page Rank 值倒序排列，并写入 pr_sorted.out 文件之中。前 20 高的 Page Rank 结果如下：

| 名称 | 入度 | 出度 | Page Rank |
|---|---|---|---|
| 箭头 | 45 | 5 | 0.0222668083002 |
| ← | 228676 | 2 | 0.0172861636988 |
| 维基数据 | 196429 | 14 | 0.00479873711217 |
| Unicode | 945 | 170 | 0.00390513158781 |
| 符号 | 283 | 78 | 0.00383177128121 |
| 中国 | 68798 | 2287 | 0.00129366014872 |
| 美国 | 70222 | 1741 | 0.00115649146484 |
| 学名 | 68384 | 20 | 0.00108457397504 |
| 法国 | 54791 | 629 | 0.000979740902801 |
| 市镇 | 64201 | 10 | 0.000972031267654 |
| 脊索动物门 | 29175 | 3 | 0.000839896529738 |
| 日本 | 59325 | 1462 | 0.000818811527809 |
| 植物界 | 42764 | 0 | 0.0007500696361 |
| 台湾 | 47474 | 2657 | 0.000688429705995 |
| 自动维基浏览器 | 31466 | 0 | 0.000662878000729 |
| 中华人民共和国 | 33233 | 3905 | 0.000660764666385 |
| 英国 | 31105 | 660 | 0.000616594610762 |
| 香港 | 45962 | 2376 | 0.000581452559999 |
| * | 13990 | 3 | 0.00055635308354 |
| 市镇 (法国) | 30974 | 58 | 0.000544901652058 |

Table 4.1: 前 20 名 Page Rank 结果

从结果中，我们发现，Page Rank 最高的竟然是“箭头”这个网页，而第二高的是“←”这个符号。原因很简单，“←”这个符号共有 20 万的入度，却只有极少的出度，其中一个出度就是“箭头”这个网页。而“Unicode”和“符号”入度非常之低，但 Page Rank 之所以这么高，也是因为“箭头”有输出指向了“Unicode”和“符号”网页。而至于“中国”、“美国”、“学名”、“法国”等词条，就是因为他们的入度比较高，所以排名也比较靠前。

总之，我们可以发现，Page Rank 要比较靠前需要满足以下两个条件之一：有较大的入度（很多网站都链接到该网站）或者被某个非常重要的网站“赏识”（投票权重较大）。

## 4.4 PageRank 结果分布情况

接着，我统计了 Page Rank 结果的分布情况，绘制如下图表：

其中横轴为倒序排布后网站编号，纵轴为 Page Rank 值的以 10 为底的对数值。从图中我们可以发现，Page Rank 值的衰减是非常非常快的，仅仅是前 10% 的网站的 Page Rank 就已经从最高值 0.02 左右降低到 $10^{-6}$，如果我们只画出前 10000 个和前 1000 个网站的 Page Rank 分布，就可以得到如下图表：

由此可见 Page Rank 衰减的速度非常之快。

### 4.5 PageRank 与入链接数的关联分析

从表4.1中我们就已经可以看出，随着 Page Rank 值的减少，入链接数并没有简简单单地随之减少，而是有较大差异。正如刚刚分析的那样，“Unicode”和“符号”入度很低，但是 Page Rank 值非常高，这是因为其由“箭头”这个 Page Rank 值很高的网页所指向而导致的。

如果画出 Page Rank 和入链接数的散点图，我们可以得到如下结果：

其中横轴是入链接数以 10 为底的对数坐标，纵轴是 Page Rank 以 10 为底的对数坐标。从图中我们可以发现，尽管二者并不是严格的正相关，但是随着入度的增加，Page Rank 是有增加的趋势的。

### 4.6 PageRank 得分与相应条目语义内容的分析

我们依次打开前几名对应的维基百科页面，可以看出“箭头”、“←”、“维基数据”、“Unicode”、“符号”这前 5 名的维基百科内容较为简单，即不应该认为其语义内容优秀。而“中国”、“美国”、“法国”、“日本”等词条则内容较为丰富和优秀。这就说明了，Page Rank 算法所反映出来的链接结构关系并不是评定一个网页好坏的唯一依据，我们依然要通过语义内容的角度进行分析，剔除一些 Page Rank 较高但却不合理的劣质网页。

## 5 实验总结

通过这次实验，我学习掌握了 Page Rank 算法的基本思想和其实现方法，并对其依据和动机有了深入的了解。另外，通过分析网站的链接结构关系，我也理解了链接结构中所说的幂率和 Bow-tie 结构。

总之，通过亲身实践，我对搜索引擎中的链接结构子系统又有了更深的理解和体会。