校内网站检索系统实现

——我的搜索引擎 Ding

Author 刘子晨

ID 1610914

Teacher 温延龙

Date 13^{th} / Dec / 2018

目录

Contents i

Abstract ii

1 引言 1

2 前期准备工作 2
2.1 命名我的搜索引擎 2
2.2 选取检索网站 2
2.3 实现获取内容爬虫 3
2.4 实现获取链接关系爬虫 6

3 链接分析——PageRank 算法实现 7
3.1 PageRank 基础算法 7
3.2 PageRank 代码实现 9

4 利用 Whoosh 构建索引 10
4.1 定义 Schema 10
4.2 构建索引 11

5 搜索及结果处理 12

6 成果展示 14

7 有趣的 PageRank 16

8 总结 18

Bibliography 19

Abstract

本实验在《简易信息检索系统实现——通过倒排索引实现布尔检索》 [1] 和《简易信息检索系统实现——通过向量空间模型实现查询排序》 [2] 的基础上进行。

本次实验中，我实现了对于南开大学校内网站的检索功能，检索网站包括：南开大学文学院、南开大学软件之家、南开大学物理学院、南开大学法学院、南开大学数学院。总计超过 11000 条网站数据，提供主题检索、网页快照、语音识别等多种功能。信息获取包括 deep web 中的网页。

搜索引擎的索引构建主要使用 whoosh，链接分析使用 PageRank 算法，采用一定的权重比例将两个评分进行合并最终按照权重排名返回结果。

本实验数据库采用 MySQL，后端使用 Python 的 Web 框架 Bottle，前端使用 Vue.js 框架进行展示。

线上地址为：http://47.95.200.233/search-homework/

你所需要的源码、介绍、展示、实验数据，均可在线上地址进行查看和下载。

Key Words: search engine，whoosh, PageRank

Chapter 1

引言

实现本次的校内信息搜索引擎首先需要获取校内网站的信息，这就需要我们实现一个校内网站的爬虫。由于本次实验需要获取 deep web 的信息，所以我们的爬虫需要从校内网站的搜索框进行检索，从而获取内容网页的网址。而后通过内容网页获取的内容进行分析，再将分词后的结果进行检索，从而获取更多的网页。

另一方面，我们的检索系统还需要实现链接分析，并且进行链接分析的算法为 PageRank，这就需要我们将整个网站中的所有表层内容爬取后才能真正实现，所以我们的爬虫还需要将网站的每个表层页面都进行爬取，并获取他们之间的链接关系。

这两个对于爬虫的要求是有点矛盾的，所以我最终决定实现两个爬虫，一个爬虫获取需要搜索的内容网页的信息，而另一个爬虫爬取网站中的链接关系和锚文本，以供实现链接分析算法 PageRank。

在获取信息并通过 PageRank 计算获取链接权重之后，我们还需要进行网页内容的索引建立，并通过建立的索引进行后期搜索，这里由于我们之前建立的后端采用的是 python 语言，所以我们选择使用 python 中知名的 Whoosh 搜索组件来进行索引的构建和内容检索。

构建索引之后，在检索过程中，首先将利用 whoosh 搜索出结果、并获取结果评分，而后将评分和链接权重进行加权计算，得出最终权重，进行排序后返回至前端。

Chapter 2

前期准备工作

2.1 命名我的搜索引擎

最终我为我的搜索引擎命名为 Ding。我起初打算将搜索引擎命名为谛听。谛听是地藏菩萨经案下伏着的通灵神兽，可以通过听来辨认世间万物。看过西游记的应该都对它有深刻的印象，我想表达我的搜索引擎像谛听一样可以辨认万物，准确的给用户回复（有点中二）。

但是后来又感觉这个名字不太好记，而且与如今的主流搜索引擎名没有关系，一般人很难将其与搜索引擎联系起来，于是我最终将我的搜索引擎取了一个类似微软搜索引擎必应（bing）的名字——Ding。该名字与谛听也有一定的谐音，并且感觉该名字读起来给人一种很快的感觉，所以最终就拍板决定了。

图 2.1: 我设计的 logo

2.2 选取检索网站

由实验要求，我们知道构建索引的网站需要具有以下几点要求：

- 易于爬取

  反爬虫机制不能做得很好

- 具有搜索功能

  以供获取 deep web 信息

- 链接跳转使用 a 标签

  方便获取链接关系

- 信息内容页面超过 1000 条

由以上几个条件，我最终选取了文学院、软件之家、数学院、法学院、物理学院、周恩来政府学院六个网站进行爬取，爬取完成后发现前五个网站获取的信息条数已经足够 10000 条，所以尽管已经爬取了周恩来政府学院的信息，但是没有为其构建索引。

2.3 实现获取内容爬虫

获取内容的爬虫根据不同爬取的网站写法略有不同，但是大体相似，我们这里只用物理学院进行举例说明。

在引言中已经提到，我们所需要实现的获取内容爬虫需要包含深网信息，所以我们的爬虫需要从校内网站的搜索框进行检索，从而获取内容网页的网址。而后通过内容网页获取的内容进行分析，再将分词后的结果进行检索，从而获取更多的网页。

获取内容爬虫选取的网页信息获取工具为 selenium，通过自动化控制 chrome 浏览器来获取网页信息。

```
from selenium import webdriver


class MyBrowser:
    def __init__(self):
        self.browser = webdriver.Chrome()
        self.url_link_list = []

    # 打开一个网址
    def open_url(self, url):
        self.browser.get(url)
        res = self.browser.current_url
        return res
```

通过分析网页标签和特征获取信息页面的标题和文章内容（该模块需要根据不同的网站的信息页面设置进行设定，所以对于爬取不同网站，该函数需要重写）：

```
    def get_content(self):
```

```
        title_text = ""
        post_time_text = ""
        content_text = ""
        try:
            info_list = self.browser.find_elements_by_xpath("//span[
                                                @class='Article_Title']
                                                ")
            title_text = info_list[0].text
            post_time = self.browser.find_element_by_xpath("//table[
                                                @class='border2']")
            post_time_text = post_time.text
            content = self.browser.find_element_by_class_name("
                                                Article_Content")
            content_text = content.text
        except Exception as e:
            print(e)
        return title_text, post_time_text, content_text
```

处理单个网址的网页内容，通过分析获取新的可供搜索的内容：

```
    def build_one_url(self, url):
        self.base_browser.open_url(url)
        title, post_time, content = self.base_browser.get_content()
        if title == "" or content == "":
            print(url, " error")
            return False
        # 当搜索池不足100时，向搜索池中添加种子
        if len(self.search_list) < 100:
            seeds = jieba.cut(content)
            for seed in seeds:
                if seed not in self.search_list:
                    self.search_list.append(seed)
        html = self.base_browser.get_html()
        # 将数据保存到数据库
        Physics.insert_data(url, title, post_time, content, html)
        return True
```

从搜索页面获取内容网页的网址（注意，并非搜索页面的所有 a 标签都是内容网页的网址，需要判断 a 标签出现的位置）：

```
# 获取搜索页面的所有连接
def get_search_urls(self, before_list):
    try:
        search_info = self.browser.find_element_by_class_name("
                                            results_list")
```

```
        search_list = search_info.find_elements_by_xpath(".//a")
        for url in search_list:
            href = url.get_attribute("href")
            # 只填加之前没有的url
            if href not in before_list:
                before_list.append(href)
    except Exception as e:
        print(e)
```

在搜索页面点击下一页：

```
# 点击下一页，有则返回true，否则返回false
def get_next_page(self):
    try:
        self.browser.find_element_by_link_text("下一页").click()
        return True
    except Exception as e:
        print(e)
        return False
```

运行时的主体函数：

```
def start_build(self):
    # 最多4000条，最多搜索100次
    while self.used_url_num < 4000 and self.search_time < 100:
        for url in self.url_list[self.used_url_num:]:
            try:
                self.build_one_url(url)
            except Exception as e:
                print(e)
            # 设置延迟，礼貌爬取
            time.sleep(3)
        # 当 当前url不足时，调用搜索功能，添加url
        try:
            self.used_url_num = len(self.url_list)
            self.base_browser.search(self.search_list[self.search_time]
                                                    )
            self.search_time += 1
            while True:
                self.base_browser.get_search_urls(self.url_list)
                # 设置延迟，礼貌爬取
                time.sleep(3)
                if not self.base_browser.get_next_page():
                    break
            self.save()
```

```
23            except Exception as e:
24                print(e)
```

2.4 实现获取链接关系爬虫

获取链接关系的爬虫相对简单，只需要将当前页面的所有 a 标签中的信息进行获取，但是我们需要对获取的 url 进行处理，将链接到外站的 url 剔除，并且需要将 url 中的一些无意义符号去除，防止爬取重复内容，该功能可以通过正则表达式和 parse 包实现。

除此之外，需要处理的便是爬取深度的问题，有时，有些网站有些网页是循环的，比如日历，这就需要我们设置一个爬取的最大深度，来跳出这些循环网页。

网页之间的链接关系保存的格式为一个列表，列表中每一项又是一个列表，小列表中第一项是出射网页，第二项是入射网页。

该部分与上部分其实类似，不再重复贴代码。

Chapter 3

链接分析——PageRank 算法实现

3.1 PageRank 基础算法

PageRank 是用于网页排序的算法，该算法起初思想的来源是学术界评价学术论文重要性的通用方法——看论文的引用次数，由此想到网页的重要性也可以通过类似的方法来进行评价。在这一思想的基础上，PageRank 的核心理念诞生了：

- 如果一个网页被很多其他网页链接到的话说明这个网页比较重要，也就是 PageRank 值会相对较高
- 如果一个 PageRank 值很高的网页链接到一个其他的网页，那么被链接到的网页的 PageRank 值会相应地因此而提高

下图是上述思想的一个直观概念图：

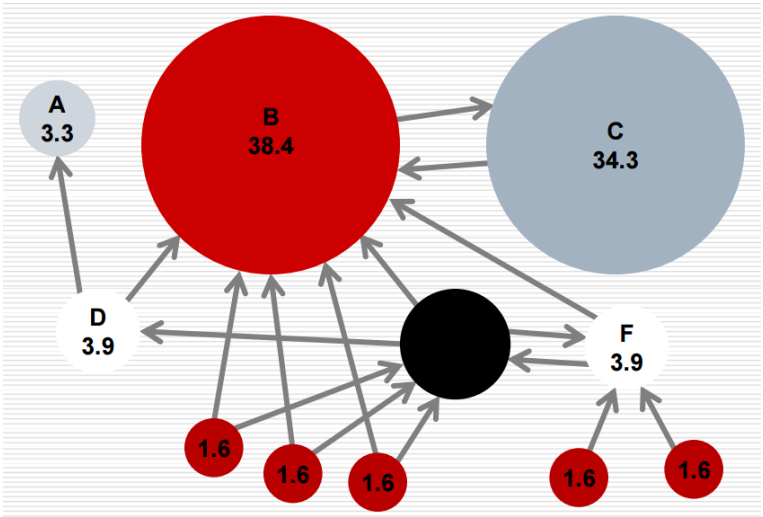

图 3.1: 一个网页的重要程度与链接到它的网页有关

PageRank 算法总的来说就是预先给每个网页一个 PR 值（下面用 PR 值指代 PageRank 值），由于 PR 值物理意义上为一个网页被访问概率，所以一般是 1/N，其中 N 为网页总数。

每个链接的投票（vote）与其源页面的 pr 值成正比。即如果一个网页的 pr 值为 r 有 n 个出度，则每个链接的投票为 r/n。同时，一个页面的 pr 值也来源于入度其的页面的投票和。

我们可以简单举例说明一下上述算法。

现有如下图的一个网络链接图，虽然图中的 C 页面是一个 dead-end，但是现在让我们把目光放在网页 B 上。

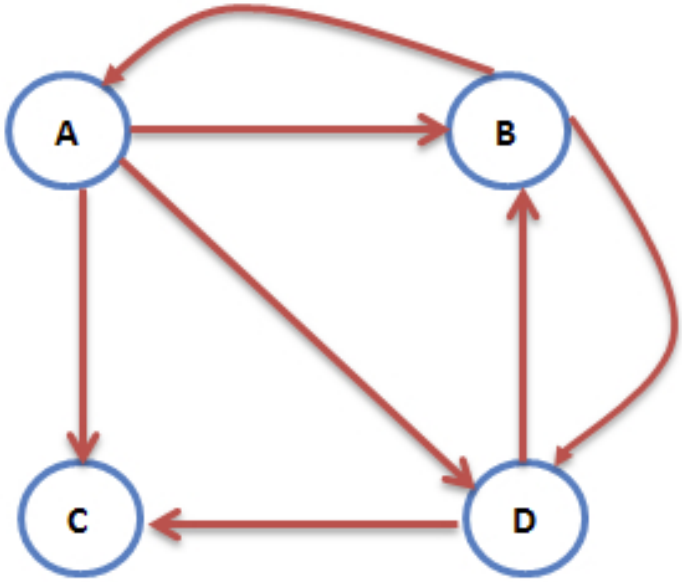

图 3.2: 一个简单的网络链接图

从图中我们可以看出，有两个链接指向了 B 节点，这两个链接分别来自 A 节点和 D 节点，A 节点有三个出度，D 节点有两个出度，从以上信息中，我们可以得到 B 节点的 PR 值:

$$PR(B) = PR(A)/3 + PR(D)/2 \tag{3.1}$$

PageRank 算法对于 PR 值的计算公式为:

$$\mathrm{PR(u)} = \mathrm{c} \sum_{v \in B_u} \frac{PR(v)}{N_v}$$

3.2 PageRank 代码实现

由于我们最后计算出来 PageRank 值是用于排序的，而且加权方式由我们自己决定，所以在计算过程中，我们只需要保证各个网址之间的权重比值是正确的即可，而不必追求权重值绝对的正确。

计算 PageRank 值所用类的成员：

```
class PageRank:
    # 阻尼系数数值
    __damping_factor = 0.85
    # 载入的url关系列表
    url_list = []
    # 存储每个url的出度
    url_count = Counter()
    # 存储每个url的权重
    url_value = Counter()
    # 全部url数量
    url_num = 0
```

因为我们之前在保存链接关系时，采用的就是 json 格式，所以读取链接关系非常简单，不再进行详细描述。

初始化处理链接关系：

```
# 初始化url的出度和初始value值
def generate_list(self):
    for u_to_u in self.url_list:
        self.url_count[u_to_u[0]] += 1
        if u_to_u[0] not in self.url_value.keys():
            self.url_value[u_to_u[0]] = 1
        if u_to_u[1] not in self.url_value.keys():
            self.url_value[u_to_u[1]] = 1
    self.url_num = len(self.url_value.keys())
```

计算 PageRank 值：

```
def page_rank(self):
    for i in range(self.times):
        new_value = Counter()
        for u_to_u in self.url_list:
            new_value[u_to_u[1]] += (self.url_value[u_to_u[0]] / self.
                                                    url_count[
                u_to_u[0]]) * self.__damping_factor
        for k in self.url_value.keys():
            new_value[k] += 1 - self.__damping_factor
        self.url_value = new_value
```

Chapter 4

利用 Whoosh 构建索引

Whoosh 是一个索引文本和搜索文本的类库，他可以为你提供搜索文本的服务，比如如果你在创建一个博客的软件，你可以用 whoosh 为它添加添加一个搜索功能以便用户来搜索博客的入口。 [3]

这一段 whoosh 官方说明文档对于 whoosh 的介绍与我们需要的索引构建需求不谋而合。

4.1 定义 Schema

开始使用 whoosh 之前，你需要一个 index 对象，在你第一次创建 index 对象时你必须定义一个 Schema 对象，Schema 对象列出了 Index 的所有域。一个域就是 Index 对象里面每个 document 的一个信息，比如他的题目或者他的内容。一个域能够被索引（就是能被搜索到）或者被存储（就是得到索引之后的结果，这对于标题之类的索引非常有用） [3]

这是 Whoosh 官方说明文档对于 Schema 的介绍，根据 Whoosh 中所给出的几个信息种类和我们的构建需求，我设计出了本实验中所需要的 Schema 对象：

```
analyzer = ChineseAnalyzer()
schema = Schema(title=TEXT(stored=True, analyzer=analyzer)
                , url=ID(stored=True, unique=True)
                , content=TEXT(stored=True, analyzer=analyzer)
                , site=ID(stored=True)
                , wid=NUMERIC(stored=True)
                , link=KEYWORD(stored=True, analyzer=analyzer)
                , pagerank=NUMERIC(stored=True))
```

对于每个需要构建索引的网页必然都有标题和内容，即 title 项和 content 项，由于这些是需要构建索引和搜索的部分，所以类型被设置为 TEXT 并为其提供中文搜索分词引擎。同时为了提供分类搜索，我们还需要保存网页的网站来源，即 site 项，该项不进行索引构建，所以类型设置为 ID。link 项为网页的锚文本，所以

较为重要，需要分配更多的权重，所以设置为 KEYWORD。wid 是文章 id，它和 PageRank 值以及 url 保存在这里只是为了方便将检索内容进行分析，否则每次检索都查询数据库，速度过慢。

4.2 构建索引

在创建好 schema 后，我们只需要按照创建的 schema，从数据库中查找出信息，而后进行索引构建即可。

```
# 插入一条索引
@staticmethod
def insert_index(title, url, content, m_type, wid,the_link, page_rank):
    # 写索引
    writer = IndexBuilder.ix.writer()
    writer.add_document(title=str(title)
                        , url=str(url)
                        , content=str(content)
                        , site=str(m_type)
                        , wid=wid
                        , link=the_link
                        , pagerank=page_rank)
    writer.commit()
```

Chapter 5

搜索及结果处理

用户通常搜索的内容都包含网页的标题和内容，而且通常出现在网页标题应该代表的权重更高，所以据此我们设计搜索条目，将网页标题或者内容中出现检索内容的都搜索出来：

```
@staticmethod
def search(keyword, type_id, page):
    searcher = DingSearch.ix.searcher(weighting=scoring.BM25F)
    m_parser = MultifieldParser(["content", "title"], IndexBuilder.
                                        schema, group=OrGroup)
    m_query = m_parser.parse(keyword)
```

在这种搜索条目的构建逻辑下，搜索“南开大学”将表示：(content: 南开 OR content: 开大 OR content: 大学 OR content: 南开大学 OR title: 南开 OR title: 开大 OR title: 大学 OR title: 南开大学)。检索内容会根据中文分词器进行分词处理。

如果用户仅对某个网站进行检索，再将搜索条件用与逻辑进行连接：

```
    if type_id in [1, 2, 3, 4, 5]:
        m_query = m_query.__and__(Term("site", str(type_id)))
```

而后进行检索，并获取全部检索条目，和用户当前搜索页的条目信息：

```
res = searcher.search(m_query)
# 得到查询总条数
res_total = len(res)
# 分页查询
res = searcher.search_page(m_query, page, 10)
```

标出搜索中匹配的内容(即为检索后在前端被特殊标红的部分)，获取评分和 PageRank 值，并重新排序后返回：

```
try:
    rr = []
    for hit in res:
```

```
        if hit.highlights("title"):
            title = hit.highlights("title")
        else:
            title = hit["title"]
        if hit.highlights("content"):
            content = hit.highlights("content")
        else:
            content = hit["content"]
        rr.append(
            {"id": hit['wid'], "url": hit['url'], "title": title, "
                                                    content": content
                , "score": hit.score, "site": hit['site'], "pagerank":
                                                        hit['pagerank']})
    return {'data': DingSearch.re_sort(rr), 'res_total': res_total}
```

根据评分和 PageRank 值重新排序的算法：

```
  @staticmethod
    def re_sort(rr):
        sort_list = []
        for v in rr:
            score = math.log10(v["pagerank"]+1) + v["score"]
            sort_list.append(SortElement(score=score, info=v))
        sort_list.sort()
        res = []
        for new_element in sort_list:
            res.append(new_element.info)
        return res
```

通过代码可以看出我的连接分析和 whoosh 评分的权重计算方法，先将 PageRank 值 +1，而后进行取对数处理，而后与 whoosh 评分相加，作为最终评分，而后排序。

Chapter 6

成果展示

在线上地址为：http://47.95.200.233/search-homework/，点击校内检索后即可进入本次作业的完成内容：

图 6.1: 校内检索页面

点击搜索后呈现的结果：


南开
综合
文学院
软件之家
数学院
法学院
物理学院
南开物理校友会筹备会召开
月20日上午，南开物理校友会筹备会在南开大学物理科学学院三教301...办的大力支持下，学院决定在近期筹备成立南开物理校友会，希望各位校友代表能积极...背景和目的，希望通过校友会的成立凝聚南开物理广大校友，为社会公益事业和南开物理学科的发展贡献力量。 校友办副
http://physics.nankai.edu.cn/_upload/article/files/3d/4c/ecb7d6a648748ffa7329da48cfd5/1ee92a12-b594
- Ding快照
南开名家讲坛”校庆日专场叶嘉莹先生谈“我与南开三十年
举行，词学大师叶嘉莹先生与现场五百多位南开校友和师生分享了自己与南开三十年的情缘。本次活动由文学院...主要学术成就。 叶嘉莹先生从南开校友张侯萍编纂并准备出版的传记《红蓼留梦...选择留在南开大学。她回顾了在南开大学创办中华古典文化研究
http://wxy.nankai.edu.cn/Article/Detail/700/65
- Ding快照
南开大学“活力南开”研究生运动会以及志愿者
发】关于举办南开大学第二届“活力南开”研究生运动会的通知.docx 关于...南开大学第二届“活力南开”研究生运动会志愿者招募
http://sms.nankai.edu.cn/_upload/article/files/40/a9/38dc44c04a8aa1e0c550de701f92/467e96bd-8302-4ae
- Ding快照

图 6.2: 结果呈现页面

Chapter 7

有趣的 PageRank

以上我们已经实现了本次的实验要求，所以这一章的内容有些类似于后记。在计算 PageRank 的过程中，我发现了一些非常有趣的事情，在这里与大家分享一下。

在通过爬取的链接关系计算得出所有页面的权重之后，我发现了各个校内网站的 PageRank 权重之间的差异性非常的大，其中物理学院的内容网页权重最高的也不到 1，而文学院网址权重最高值可达到 44，而且权重大于 1 的网页也非常多，要知道，我们这里所说的都是内容网址，而不包含首页和索引列表等网页。以下为本次爬取的所有网站的链接权重统计：

网站	PAGERANK 最大值	最小值	平均值
文学院	44.898021	0.153828	0.4111526805162
软件之家	21.150797	0.310316	1.0418487409579
法学院	1.421725	0.15	0.1997627795275
物理学院	0.901681	0.15	0.1876901514423
数学院	1.624075	0.15	0.2040505001603

图 7.1: 各个学院的 PageRank 值统计

对于这一现象我非常的困惑，于是我决定从网站本身的特点来找寻原因。

首先我查看了网页权重普遍较小的物理学院，在该学院网站的首页我发现了这样一个可能造成页面链接权重较小的罪魁祸首：

图 7.2: 南开大学物理学院网站首页

没错，就是右下角被圈出的这个 English，这个标签指向物理学院的英文版网

站，而且该 layout 是包含在所有的网址的，也就是所有的中文网址都会分一部分权重至该网址，而更为致命的是，在点进该英文版网站后，我发现，该英文版网站并没有链接指向中文版网站，这样在计算 PageRank 时就会使得整个英文版网站形成一个大的 spider trip，吸收了大量的权重，却没有放出，最终造成中文版网站的权重非常低。

法学院和数学院的情况略有不同，但是大体类似。到了这里问题的答案似乎找到了，却又有些地方不太对劲，软件之家没有英文版所以权重较高很正常，难道文学院就没有英文版吗？相信每个读者都会提出相同的问题，所以我又去文学院的首页查看了一下：

图 7.3: 南开大学文学院网站首页

结果发现文学院不但有英文版，还有繁体版，那这么说是我们之前的结论错了吗？又或是 PageRank 计算方法不对？但是当我点开了英文版之后却发现页面并没有跳转，仔细查看后，我发现三个链接都指向的是当前的首页，也就是说实际上文学院根本就没有英文版和繁体版的网站（发现这个之后真的笑死我了）。

在写爬虫和进行链接分析的过程中，其实还有很多类似的有趣的事情，让我切切实实感受到自己亲手从头至尾完成一个项目是一个非常有趣也非常有意义的过程。

Chapter 8

总结

本次作业最终回顾起来，感觉耗时很长，但是大部分时间似乎并没有用于与信息检索最为相关的索引构建和搜索结果处理上，反而更多的用在了一些前期准备如爬虫，或者后期处理，如前端绘制上。似乎这次作业整体做下来与这门课的理论似乎关系并不大。但是仔细想象，可能这才是真正在做工程时，常规的情况，有理论的部分，但是更多的工作是用在处理与工程相关的各个小问题上。

总的来说，通过这次作业，我对于爬虫的实现进步了很多，而且在爬虫过程中，由于调研了很多网站，对于网站常出的 bug 也有了一定的了解，而对于这门课的核心信息检索而言，我也算是可以说自己切实实现了一个搜索引擎了。

如今这个搜索引擎其实有很多可以优化的部分，而且如何优化我基本上实现方法都已经想好了，但是碍于这个 12 月的中旬天气格外的冷，作业格外的多，所以打算等到明年春暖花开，作业量没那么大的时候再实现了。

参考文献

[1] 刘子晨, “简易信息检索系统实现——通过倒排索引实现布尔检索,” 2018.

[2] ——, “简易信息检索系统实现——通过向量空间模型实现查询排序,” 2018.

[3] whoosh, “Whoosh quick start,” 2018, [Online; accessed 15-August-2018]. [Online]. Available: \url{https://whoosh.readthedocs.io/en/latest/quickstart.html}