



南开大学
Nankai University

计算机学院
信息检索系统原理

南开网站搜索引擎

专业：计算机科学与技术

学号：**2212032**

姓名：徐亚民

日期：**2024 年 12 月 18 日**

目录

一、 实验要求	1
1.1 网页抓取	1
1.2 文本索引	1
1.3 链接分析	1
1.4 查询服务	1
1.5 个性化查询	2
1.6 Web 界面	2
1.7 个性化推荐	2
二、 实验过程	2
2.1 网页抓取	2
2.1.1 网页抓取代码	2
2.1.2 代码功能分析	7
2.1.3 网页抓取结果	8
2.2 获取内容	9
2.2.1 获取内容代码	9
2.2.2 代码功能分析	10
2.2.3 获取内容结果	12
2.3 构建索引	12
2.3.1 构建索引代码	12
2.3.2 代码功能分析	14
2.3.3 构建索引结果	15
2.4 加载文本索引	16
2.4.1 操作实现代码	16
2.4.2 代码功能分析	16
2.5 实现链接分析	17
2.5.1 操作实现代码	17
2.5.2 代码功能分析	18
2.6 向量空间模型	19
2.6.1 操作实现代码	19
2.6.2 代码功能分析	20

2.7	用户系统实现	21
2.7.1	操作实现代码	21
2.7.2	代码功能分析	22
2.8	其余部分实现	22
2.8.1	操作实现代码	22
2.8.2	代码功能分析	25
三、	实验结果	27
3.1	文档查询截图	27
3.2	短语查询截图	27
3.3	通配查询截图	28
3.4	网页快照截图	28
3.5	其余功能截图	29
四、	实验总结	29
4.1	网页抓取	29
4.2	文本索引与链接分析	30
4.3	查询服务	30
4.4	个性化查询与推荐	30
4.5	Web 界面与交互	31
4.6	挑战与改进空间	31

一、 实验要求

本次作业针对南开校内资源构建 Web 搜索引擎，为用户提供南开信息的查询服务和个性化推荐。作业为半开放性题目，你可以只针对某一方面的资源构建搜索主题，如南开动漫资源站，南开新闻资源站等。也可以制作综合性的资源搜索平台，类似 Baidu、Google 等。

具体实现细节不做要求，自己制定主题，但请至少包含本手册中涉及到的模块。作业可以借助各种工具和包，请大家善用并减少重复工作量。推荐采用 ElasticSearch。

本次作业主要包含网页抓取、文本索引、链接分析、查询服务、个性化查询、Web 界面、个性化推荐七个任务。

1.1 网页抓取

对南开校内资源进行抓取，依据自己主题而定。抓取网页数量至少为 10 万，低于该数量将酌情扣分。再次强调，请在校内抓取、礼貌抓取，遵循爬虫协议。

1.2 文本索引

对网页及其锚文本构建索引，可以设置多个索引域，比如：网页标题，URL，锚文本等。

1.3 链接分析

使用 PageRank 进行链接分析，评估网页权重。

1.4 查询服务

基于向量空间模型，结合链接分析结果对查询结果进行排序。为用户提供站内查询、文档查询、短语查询、通配查询、查询日志、网页快照等共计六种。

- 站内查询：基本的查询操作，送分项。
- 文档查询：一些网页可能会携带或其本身就是附件下载链接，支持对文档（doc/docx, pdf, xls/xlsx 等）的查询操作。

- 短语查询：也是基本的查询操作，支持对多个 Term 的查询。注意，“南开 1 大学 2”和“南开 1 是一所综合性大学 2”的区别。
- 通配查询：用户可能并不知道自己要查什么，支持通配符（正则）查询操作。例如：代表多个字符，? 代表一个字符，“温”用于查询所有姓温的老师，“计?”用于查询如“计网”、“计算”等词。
- 查询日志：参考百度的查询日志（历史），能够知道用户之前查了什么。
- 网页快照：网页快照是搜索引擎在收录网页时，对网页进行备份，存在自己的服务器缓存里，当用户在搜索引擎中点击链接时，搜索引擎将爬虫系统当时所抓取并保存的网页内容展现出来，称为网页快照。

1.5 个性化查询

针对不同登录用户提供不同的内容排序。建议实现一个注册/登录系统，通过提供不同的用户信息实现内容排序算法的修正。

1.6 Web 界面

本次作业不强调界面美观度，以功能为主。因此，可以 Terminal，可以 WebPage，二者不会存在得分区别。

1.7 个性化推荐

个性化推荐存在两类，一个是搜索上的联想关联，一个是内容分析后的推荐。任选其中一种实现即可。

二、 实验过程

2.1 网页抓取

2.1.1 网页抓取代码

```
1 # 验证 Elasticsearch 服务已启动
2
3 from elasticsearch import Elasticsearch
```

```
4
5 # 添加 'scheme' 参数, 指定连接协议
6 es = Elasticsearch([{'host': 'localhost', 'port': 9200, 'scheme': 'http'}])
7
8 # 检查 Elasticsearch 服务是否运行
9 if not es.ping():
10     print("Elasticsearch 服务未响应!")
11 else:
12     print("Elasticsearch 服务已启动!")
13
14 # 网页爬取
15 import requests
16 from bs4 import BeautifulSoup
17 import time
18 import random
19 import re
20 import os
21 import pickle
22 import csv
23 from urllib.parse import urljoin
24 import hashlib # 用于计算 URL 的哈希值
25 import chardet # 用于自动检测网页编码
26
27 # 设置爬虫基本参数
28 BASE_URL = "http://www.nankai.edu.cn/" # 南开大学官网
29 MAX_PAGES = 100120
30 DELAY = 0.001
31 USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
    Gecko) Chrome/58.0.3029.110 Safari/537.36"
32 HEADERS = {"User-Agent": USER_AGENT}
33
34 # 设置文件存储路径
35 SAVE_DIR = "HtmlFile"
36 if not os.path.exists(SAVE_DIR):
37     os.makedirs(SAVE_DIR)
38
39 # 存储爬虫状态的文件路径
40 STATE_FILE = "crawling_state.pkl"
41 # 存储网页内容的 CSV 文件路径
42 CSV_FILE = "Html_File.csv"
43
44 def fetch_page(url):
45     try:
46         response = requests.get(url, headers=HEADERS, timeout=5)
47         if response.status_code == 200:
48             # 检测页面的编码
49             encoding = response.apparent_encoding # apparent_encoding会自动推断编码
50             response.encoding = encoding # 根据推断的编码设置正确的编码
51             return response.text
52         else:
53             print(f"Failed to retrieve {url} with status code {response.status_code}")
54             return None
55     except requests.exceptions.RequestException as e:
56         print(f"Error fetching {url}: {e}")
57         return None
58
```

```
59 def parse_page(url, html):
60     soup = BeautifulSoup(html, "html.parser")
61
62     # 提取网页标题
63     title = soup.title.string if soup.title else "No Title"
64     if not title: # 如果没有标题, 尝试从 meta 标签中获取
65         meta_title = soup.find("meta", {"name": "description"})
66         if meta_title:
67             title = meta_title.get("content", "No Title")
68         else:
69             title = "No Title"
70
71     # 清理标题, 避免包含非法字符
72     title = clean_filename(title)
73
74     # 保存网页内容到本地文件
75     save_page_to_file(url, html, title)
76
77     return {
78         "url": url,
79         "title": title,
80         "timestamp": time.time(),
81     }
82
83 # 保留的 Windows 文件名
84 RESERVED_NAMES = {
85     "CON", "PRN", "AUX", "NUL",
86     "COM1", "COM2", "COM3", "COM4", "COM5", "COM6", "COM7", "COM8", "COM9",
87     "LPT1", "LPT2", "LPT3", "LPT4", "LPT5", "LPT6", "LPT7", "LPT8", "LPT9"
88 }
89
90 def clean_filename(filename):
91     # 如果 filename 是 None 或空字符串, 返回一个默认名称
92     if not filename:
93         return "untitled_page"
94
95     # 限制文件名长度为 100 个字符
96     if len(filename) > 100:
97         filename = filename[:100]
98
99     # 去除开头和结尾的空格
100     filename = filename.strip()
101
102     # 如果 filename 不是 None 类型, 则执行编码和清理操作
103     filename = filename.replace("\r", "").replace("\t", "").replace("\n", "") # 去掉换
104     # 行符
105     filename = re.sub(r'[\./:*?"<>|]', '', filename) # 替换非法字符
106
107     # 检查文件名是否为保留的名称
108     if filename.upper() in RESERVED_NAMES:
109         filename = filename + "_page"
110
111     # 去除文件名末尾的点 and 空格
112     filename = filename.rstrip(" .")
113
114     return filename
```

```
114
115 def save_page_to_file(url, html, title):
116
117     # 创建文件路径
118     file_name = f"{title}.html"
119     file_path = os.path.join(SAVE_DIR, file_name)
120
121     # 如果文件已经存在, 添加时间戳后缀
122     if os.path.exists(file_path):
123         timestamp = int(time.time()) # 获取当前时间戳
124         file_name = f"{title}_{timestamp}.html"
125         file_path = os.path.join(SAVE_DIR, file_name)
126
127     # 保存网页 HTML 内容到文件
128     with open(file_path, 'w', encoding='utf-8') as file:
129         file.write(html)
130     print(f"Page saved to {file_path}")
131     return True # 返回 True, 表示文件已成功保存
132
133 def extract_links(url, html):
134     soup = BeautifulSoup(html, "html.parser")
135     links = set()
136
137     # 提取所有的 <a> 标签, 并获取其中的 href 属性
138     for a_tag in soup.find_all("a", href=True):
139         link = a_tag["href"]
140
141         # 处理相对链接和绝对链接
142         if link.startswith("http"):
143             links.add(link)
144         else:
145             # 使用 urljoin 处理相对链接
146             full_link = urljoin(url, link)
147
148             # 过滤掉一些不需要的链接, 如锚点 (#) 或空链接
149             if not full_link.endswith("#") and full_link != url:
150                 links.add(full_link)
151
152     return links
153
154 def save_state(to_crawl, crawled):
155     # 保存当前的抓取状态到文件
156     with open(STATE_FILE, 'wb') as state_file:
157         pickle.dump((to_crawl, crawled), state_file)
158     print(f"State saved to {STATE_FILE}")
159
160 def load_state():
161     # 从文件加载之前保存的爬虫状态
162     if os.path.exists(STATE_FILE):
163         with open(STATE_FILE, 'rb') as state_file:
164             to_crawl, crawled = pickle.load(state_file)
165             print(f"State loaded from {STATE_FILE}")
166             return to_crawl, crawled
167     else:
168         return set(), set()
169
```



```
170 def save_to_csv(data):
171     # 保存网页信息到 CSV 文件
172     file_exists = os.path.exists(CSV_FILE)
173
174     with open(CSV_FILE, 'a', encoding='utf-8', newline='') as f:
175         writer = csv.writer(f)
176         if not file_exists:
177             # 写入标题行
178             writer.writerow(["title", "url"])
179             writer.writerow([data["title"], data["url"]])
180         print(f"Page data saved to {CSV_FILE}")
181
182 def get_url_hash(url):
183     # 确保 url 是字符串类型
184     url = str(url)
185     # 使用 hashlib 计算 URL 的哈希值
186     return hashlib.md5(url.encode('utf-8')).hexdigest()
187
188 def crawl(start_url):
189     # 尝试从文件加载之前的状态
190     to_crawl, crawled = load_state()
191
192     # 如果起始 URL 不在待抓取列表中, 加入待抓取列表
193     to_crawl.add(start_url)
194
195     while to_crawl and len(crawled) < MAX_PAGES:
196         url = to_crawl.pop()
197
198         # 获取 URL 的哈希值
199         url_hash = get_url_hash(url)
200
201         # 如果该 URL 的哈希值已存在于 crawled 集合中, 跳过
202         if url_hash in crawled:
203             continue
204
205         # 如果 URL 后缀是 .mp4, 则跳过这个 URL
206         if url.endswith('.mp4'):
207             print(f"Skipping: {url} (MP4 file)")
208             continue
209
210         # 如果 URL 后缀是 .png, 则跳过这个 URL
211         if url.endswith('.png'):
212             print(f"Skipping: {url} (PNG file)")
213             continue
214
215         # 如果 URL 后缀是 .jpg, 则跳过这个 URL
216         if url.endswith('.jpg'):
217             print(f"Skipping: {url} (JPG file)")
218             continue
219
220         # 如果 URL 后缀是 .pdf, 则跳过这个 URL
221         if url.endswith('.pdf'):
222             print(f"Skipping: {url} (PDF file)")
223             continue
224
225         print(f"Crawling: {url}")
```

```
226     html = fetch_page(url)
227
228     if html:
229         document = parse_page(url, html)
230
231         # 保存页面数据到 CSV 文件
232         save_to_csv(document)
233         # 文件成功保存, 则更新状态
234         crawled.add(url_hash) # 使用 URL 的哈希值进行存储
235         # 提取新的链接
236         new_links = extract_links(url, html)
237         # 更新待爬取链接集合
238         to_crawl.update(new_links)
239
240         # 每次抓取后保存状态
241         save_state(to_crawl, crawled)
242
243         # 随机延迟
244         time.sleep(DELAY + random.uniform(0.000, 0.009))
245
246 if __name__ == "__main__":
247     crawl(BASE_URL) # 从 BASE_URL 开始抓取
```

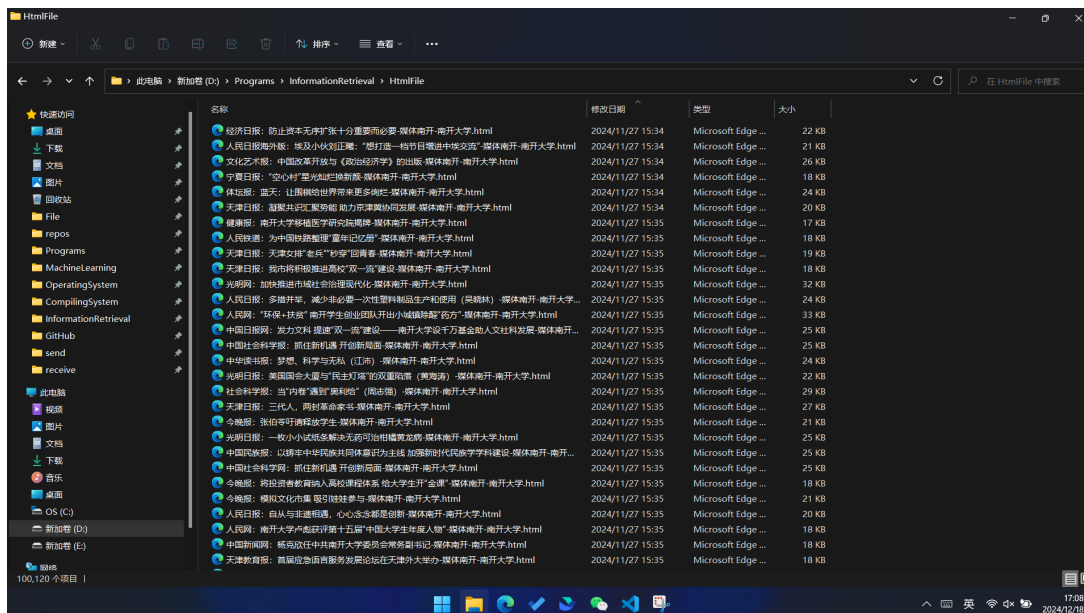
2.1.2 代码功能分析

- 本代码主要由两部分功能组成：首先是验证 Elasticsearch 服务是否已经启动，其次是实现网页爬取与存储功能。代码通过 **Elasticsearch** 模块连接本地的 Elasticsearch 服务，并通过 **ping** 方法检查其是否响应。
- 通过 **Elasticsearch({'host': 'localhost', 'port': 9200, 'scheme': 'http'})** 创建 Elasticsearch 客户端并检查服务状态。如果服务未响应，则输出“Elasticsearch 服务未响应!”，否则输出“Elasticsearch 服务已启动!”。
- 网页爬取部分使用了 **requests** 模块获取网页内容，**BeautifulSoup** 用于解析 HTML 内容。爬虫从指定的 **BASE_URL** 开始，获取网页并解析其中的链接。
- 在爬取过程中，通过 **fetch_page** 函数向指定 URL 发送请求并获取 HTML 内容。如果请求成功且返回状态码为 200，便返回网页内容，否则输出错误信息并返回 **None**。
- 解析网页时，首先从网页中提取标题，若标题不存在则尝试从 **meta** 标签中获取。提取的标题经过 **clean_filename** 函数处理，确保其符合文件命名规范，并去除非法字符。
- 网页内容通过 **save_page_to_file** 函数保存为本地文件，保存路径由标题决定，并确保文件名不与 Windows 保留名称冲突。若文件已存在，则在文件名后添加时间

戳以避免覆盖。

- 通过 **extract_links** 函数从 HTML 内容中提取所有的链接，并通过 **urljoin** 处理相对链接，将它们转换为绝对链接。函数返回所有有效链接的集合。
- 代码还包括状态保存与加载功能，使用 **pickle** 模块将当前爬取状态（待爬取的 URL 集合和已爬取的 URL 集合）保存到文件中，并在爬虫重新启动时加载先前的状态。
- 网页信息还会被保存到 CSV 文件中，函数 **save_to_csv** 负责将爬取到的网页标题和 URL 存入 CSV 文件中。文件会追加写入数据，首次写入时会添加表头。
- **get_url_hash** 函数使用 **hashlib** 模块生成 URL 的哈希值，用于唯一标识 URL 并避免重复爬取。爬虫在每次抓取后会更新爬取状态，确保不会重复抓取相同的页面。
- 爬虫在抓取过程中会随机延迟，减少对目标服务器的压力，并且会跳过不需要的文件类型（如 .mp4, .png, .jpg, .pdf 等）。
- **crawl** 函数是爬虫的主循环，负责控制爬虫的抓取流程。它会持续抓取网页，解析内容，提取链接，更新状态并随机延迟。循环直到达到最大页面数或没有更多页面可抓取。

2.1.3 网页抓取结果



名称	修改日期	类型	大小
经济日报: 防止资本无序扩张十分重要而必要-媒体南开-南开大学.html	2024/11/27 15:34	Microsoft Edge ...	22 KB
人民日报海外版: 埃及小议到正策: "想打造一特节目增进中埃交流"-媒体南开-南开大学.html	2024/11/27 15:34	Microsoft Edge ...	21 KB
文艺艺不报: 中国改革开放与《政治经济学》的出版-媒体南开-南开大学.html	2024/11/27 15:34	Microsoft Edge ...	26 KB
宁夏日报: "空心村"星光灿烂新颜-媒体南开-南开大学.html	2024/11/27 15:34	Microsoft Edge ...	18 KB
体坛报: 蓝天: 让重钢给世界带来更多绚烂-媒体南开-南开大学.html	2024/11/27 15:34	Microsoft Edge ...	24 KB
天津日报: 凝聚共识汇聚力量 助力京津冀协同发展-媒体南开-南开大学.html	2024/11/27 15:34	Microsoft Edge ...	20 KB
健康报: 南开大学移植医学研究取得新进展-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	17 KB
人民日报: 为中国科技创新"量"中化"质"-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	18 KB
天津日报: 天津女队"备战"中考"备战"季-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	18 KB
天津日报: 向志将给花迎雨双-一"指"建设-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	18 KB
光明网: 加快推进市域社会治理现代化-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	32 KB
人民日报: 多措并举, 减少非必要一次性塑料制品生产和使用(吴晓林)-媒体南开-南开大学...	2024/11/27 15:35	Microsoft Edge ...	24 KB
人民日报: "环良+扶老" 南开学生创业团队开出"小微企业"助力-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	33 KB
中国日报网: 助力文科 提振"双一"建设——南开大学设千万基金助力人文社科发展-媒体南开...	2024/11/27 15:35	Microsoft Edge ...	25 KB
中国社会科学报: 抓住新机遇 开创新局面-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	25 KB
中华书局: 智慧、科学和无私(江涛)-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	24 KB
光明日报: 美国国会大厦与"民主灯塔"的双重陷落(黄海清)-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	22 KB
社会科学报: 当"内容"遇到"便利" (周志德)-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	29 KB
天津日报: 三代人, 两封革命家书-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	27 KB
今晚报: 张伯苓吁请解放学生-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	21 KB
光明日报: 一枚小小试纸多解决无药可治相繼龍-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	25 KB
中国新闻网: 以铸牢中华民族共同体意识为主线 加强新时代民族事务学科建设-媒体南开-南开...	2024/11/27 15:35	Microsoft Edge ...	25 KB
中国社会科学报: 抓住新机遇 开创新局面-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	25 KB
今晚报: 将投资者教育纳入高校课程体系 给大学生开"金课"-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	18 KB
今晚报: 模拟文化市集 吸引年轻参与-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	21 KB
人民日报: 自从与非遗相遇, 心念念都是创新-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	20 KB
人民日报: 南开大学入选评第十五届"中国大学生年度人物"-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	18 KB
中国新闻网: 杨克勤任中共南开大学委员会常务副书记-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	18 KB
天津教育报: 首届应急志愿服务论坛在天津大学举办-媒体南开-南开大学.html	2024/11/27 15:35	Microsoft Edge ...	18 KB

图 1: 网页抓取结果

2.2 获取内容

2.2.1 获取内容代码

```
1 import os
2 import csv
3 import re
4 from bs4 import BeautifulSoup
5
6 # 假设html文件夹路径
7 html_folder_path = './Html_File'
8
9 # 读取Html_File.csv并获取html文件名称和URL
10 def read_csv(csv_file):
11     with open(csv_file, mode='r', encoding='utf-8') as file:
12         reader = csv.reader(file)
13         next(reader) # 跳过表头
14         file_data = [row for row in reader]
15     return file_data
16
17 # 提取html文件中的信息：标题、URL、锚文本、正文
18 def extract_html_info(html_file, url, html_file_name):
19     with open(html_file, 'r', encoding='utf-8') as file:
20         soup = BeautifulSoup(file, 'html.parser')
21
22         # 直接使用文件名作为标题
23         title = html_file_name # 这里将title设置为文件名
24
25         # 提取正文，通常为 <body> 标签中的内容
26         body = soup.find('body')
27         body_text = body.get_text(strip=True) if body else '无正文'
28
29         # 格式化正文，去除换行符、制表符等
30         formatted_body = format_text(body_text)
31
32         # 提取锚文本 (<a>标签的文本)
33         anchors = soup.find_all('a')
34         anchor_texts = [a.get_text(strip=True) for a in anchors]
35
36         # 返回提取的信息
37         return {
38             'title': title,
39             'url': url,
40             'anchors': anchor_texts,
41             'body': formatted_body
42         }
43
44 # 格式化文本：去除多余的空格、换行符、制表符等
45 def format_text(text):
46     # 使用正则表达式去除多余的空白字符（换行、制表符、多个空格等）
47     text = re.sub(r'\s+', ' ', text) # 替换所有类型的空白字符（如换行、空格等）为单个
        空格
48     text = text.strip() # 去掉文本开头和结尾的空白字符
49     return text
50
```

```
51 # 输出网页内容并写入到新的CSV文件
52 def get_content_and_write_to_csv(file_data, output_csv_file):
53     # 创建并打开CSV文件进行写入
54     with open(output_csv_file, mode='w', encoding='utf-8', newline='') as csvfile:
55         fieldnames = ['title', 'url', 'anchors', 'body']
56         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
57
58         writer.writeheader() # 写入表头
59
60         # 遍历每一行文件数据, 提取HTML信息并写入CSV文件
61         for row in file_data:
62             html_file_name = row[0] # 获取html文件名
63             url = row[1] # 获取url
64             html_file_path = os.path.join(html_folder_path, f"{html_file_name}.html")
65
66             if os.path.exists(html_file_path):
67                 html_info = extract_html_info(html_file_path, url, html_file_name)
68                 writer.writerow(html_info)
69             else:
70                 print(f"Warning: HTML file {html_file_name}.html not found.")
71
72         print(f"Index successfully written to {output_csv_file}")
73
74 # 主函数
75 def main():
76     input_csv = './Html_File.csv' # 输入网页文件CSV文件路径
77     output_csv = './Html_Content.csv' # 输出网页内容CSV文件路径
78
79     file_data = read_csv(input_csv) # 读取原始HTML文件名和URL数据
80     get_content_and_write_to_csv(file_data, output_csv) # 输出网页内容并且写入CSV文件
81
82 if __name__ == "__main__":
83     main()
```

2.2.2 代码功能分析

- **read_csv** 函数: 该函数用于读取一个 CSV 文件, 其中包含 HTML 文件名及其对应的 URL。它使用 Python 内置的 `csv.reader` 模块读取文件内容, 并将每行数据存储在 **file_data** 列表中, 返回这个列表。注意: 第一行表头被跳过。
- **extract_html_info** 函数: 该函数用于从指定的 HTML 文件中提取关键信息。其主要任务包括:
 - 使用 **BeautifulSoup** 库解析 HTML 文件内容。
 - 将 HTML 文件的文件名作为网页的标题 (**title**)。
 - 提取网页正文 (**body**), 并通过 **format_text** 函数进行格式化, 去除不必要的空白字符。

- 提取所有锚文本 (**anchors**) 数据, 即网页中所有 `<a>` 标签的文本。

函数返回一个包含标题、URL、锚文本和正文的字典。

- **format_text** 函数: 此函数用于格式化提取的网页正文。它使用正则表达式去除文本中的多余空白字符 (如换行符、制表符及多个空格), 并返回经过格式化的正文文本。
- **get_content_and_write_to_csv** 函数: 此函数用于将提取的网页内容 (包括标题、URL、正文和锚文本) 写入一个新的 CSV 文件。具体步骤如下:
 - 使用 `csv.DictWriter` 创建 CSV 文件并定义列名 (`title`、`url`、`anchors`、`body`)。
 - 遍历传入的文件数据 **file_data**, 依次读取每个 HTML 文件的路径, 调用 **extract_html_info** 函数提取网页内容, 最后将提取的数据写入 CSV 文件。
 - 如果指定的 HTML 文件不存在, 打印警告信息。
- **main** 函数: 该函数为程序的主入口, 负责调用其它函数进行整体处理。具体步骤如下:
 - 定义输入 CSV 文件 (`Html_File.csv`) 和输出 CSV 文件 (`Html_Content.csv`) 的路径。
 - 调用 **read_csv** 函数读取输入 CSV 文件的数据, 并存储在 **file_data** 中。
 - 调用 **get_content_and_write_to_csv** 函数, 将提取的网页信息写入输出 CSV 文件。
- **程序的总体流程**: 程序的总体流程为: 从 CSV 文件读取 HTML 文件名和 URL 数据, 提取每个 HTML 文件中的相关信息 (如标题、正文、锚文本), 然后将这些信息保存到新的 CSV 文件中, 最后完成整个数据处理过程。处理过程中, 如果遇到文件缺失, 则输出警告信息。

2.2.3 获取内容结果

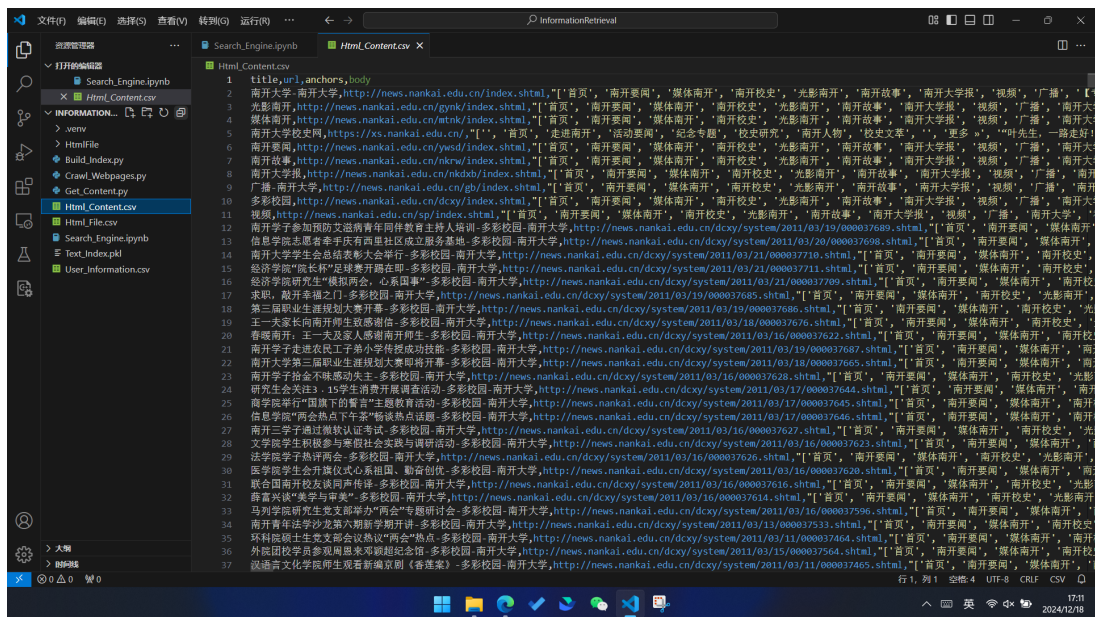


图 2: 获取内容结果

2.3 构建索引

2.3.1 构建索引代码

```
1 import jieba
2 import csv
3 import math
4 from collections import defaultdict
5 import pickle
6
7 # 增加字段大小限制, 设置为更大的值 (比如 10MB)
8 csv.field_size_limit(10000000)
9
10 # 读取 CSV 文件
11 def read_csv(file_path):
12     with open(file_path, 'r', encoding='utf-8') as f:
13         reader = csv.DictReader(f)
14         return [row for row in reader]
15
16 # 中文分词
17 def tokenize(text):
18     return list(jieba.cut(text))
19
20 # 计算文档的 TF
21 def compute_tf(doc):
22     tf = defaultdict(int)
23     for word in doc:
24         tf[word] += 1
25     # 将词频转化为 TF
```

```
26     total_words = len(doc)
27     for word in tf:
28         tf[word] /= total_words
29     return tf
30
31 # 计算所有文档的IDF
32 def compute_idf(documents):
33     idf = defaultdict(int)
34     total_docs = len(documents)
35
36     for doc in documents:
37         unique_words = set(doc)
38         for word in unique_words:
39             idf[word] += 1
40
41 # 计算IDF
42 for word in idf:
43     idf[word] = math.log(total_docs / (1 + idf[word])) # 加1防止除零
44
45     return idf
46
47 # 构建倒排索引
48 def build_inverted_index(documents):
49     inverted_index = defaultdict(list)
50     doc_tfs = []
51
52 # 遍历文档
53 for doc_id, doc in enumerate(documents):
54     if doc_id % 500 == 0: # 每500个文档输出一行
55         print(f"正在处理第 {doc_id} 个文档...")
56     tf = compute_tf(doc)
57     doc_tfs.append(tf)
58     for word in tf:
59         inverted_index[word].append((doc_id, tf[word]))
60
61 # 计算IDF
62 idf = compute_idf(documents)
63
64 # 保存倒排索引和TF-IDF计算结果
65 return inverted_index, doc_tfs, idf
66
67 # 保存倒排索引到文件
68 def save_index(inverted_index, doc_tfs, idf, filename="Text_Index.pkl"):
69     with open(filename, 'wb') as f:
70         pickle.dump((inverted_index, doc_tfs, idf), f)
71
72 if __name__ == "__main__":
73     # 读取数据
74     print("开始读取 CSV 文件...")
75     documents = read_csv('Html_Content.csv')
76     print(f"已读取 {len(documents)} 行数据。")
77
78 # 提取每个文档的文本内容 (使用anchors和body)
79 docs = []
80 for doc_id, doc in enumerate(documents):
81     text = doc['anchors'] + " " + doc['body']
```



```
82     words = tokenize(text)
83     docs.append(words)
84
85     # 每500个文档输出一次
86     if doc_id % 500 == 0:
87         print(f"已处理 {doc_id} 个文档...")
88
89     # 构建倒排索引
90     print("开始构建倒排索引...")
91     inverted_index, doc_tfs, idf = build_inverted_index(docs)
92
93     # 保存倒排索引
94     print("开始保存倒排索引...")
95     save_index(inverted_index, doc_tfs, idf)
96     print("倒排索引构建并保存完成")
```

2.3.2 代码功能分析

- 代码首先导入了必要的库，包括 **jieba**（用于中文分词）、**csv**（用于读取 CSV 文件）、**math**（用于数学计算）、**defaultdict**（用于创建具有默认值的字典）以及 **pickle**（用于序列化对象）。
- **csv.field_size_limit(10000000)**：这行代码设置 CSV 文件的字段大小限制为 10MB，以便读取大文件时不出错。
- **read_csv(file_path)**：该函数读取指定路径的 CSV 文件，并返回文件中所有行的字典形式，每一行是一个字典，字段名是 CSV 文件的列标题。
- **tokenize(text)**：该函数使用 **jieba.cut** 对传入的中文文本进行分词，返回一个分词后的词语列表。
- **compute_tf(doc)**：该函数计算文档中每个词的词频（TF）。首先，统计每个词的出现次数，并将词频归一化为文档中总词数的比率。返回的结果是一个字典，表示每个词的词频。
- **compute_idf(documents)**：该函数计算所有文档的逆文档频率（IDF）。首先，统计每个词在多少个文档中出现。然后，计算每个词的 IDF 值，使用公式：

$$\text{IDF}(\text{word}) = \log \left(\frac{N}{1 + \text{df}(\text{word})} \right)$$

其中，**N** 是文档总数，**df(word)** 是包含该词的文档数量。加 1 是为了避免除零错误。

- **build_inverted_index(documents)**：该函数构建倒排索引。它遍历所有文档，对每

个文档计算其词频 (TF)，并将每个词及其在文档中的 TF 存储在倒排索引中。每处理 500 个文档，会打印进度信息。最后，函数还会计算所有文档的 IDF，并返回倒排索引、文档的 TF 以及 IDF。

- **save_index(inverted_index, doc_tfs, idf, filename):** 该函数将构建好的倒排索引、文档的 TF 和 IDF 保存到指定文件中，使用 **pickle.dump** 进行序列化保存。
- 在 **main** 函数中，首先通过调用 **read_csv** 函数读取 CSV 文件 `Html_Content.csv`，并提取其中每个文档的 `anchors` 和 `body` 字段作为文本内容。
- 每个文档的文本内容被传递到 **tokenize** 函数进行分词，得到词语列表，并将其添加到文档列表 **docs** 中。每处理 500 个文档，都会输出一次进度信息。
- 在读取并处理完所有文档后，调用 **build_inverted_index** 函数开始构建倒排索引，并计算文档的 TF 和 IDF。
- 最后，调用 **save_index** 函数将倒排索引、TF 和 IDF 保存到名为 `Text_Index.pkl` 的文件中，以便后续使用。
- 该程序的主要目的是通过文本分析（包括中文分词、TF-IDF 计算等），为文本数据构建一个倒排索引，并将其序列化保存，以便进行快速搜索和信息检索。

2.3.3 构建索引结果

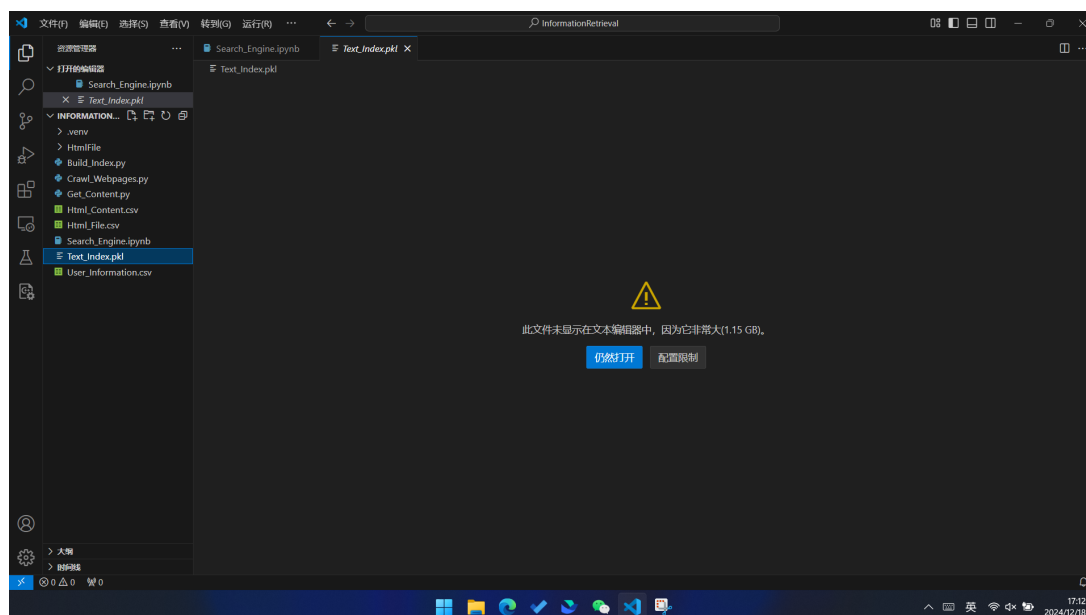


图 3: 构建索引结果

2.4 加载文本索引

2.4.1 操作实现代码

```
1 import os
2 import csv
3 import math
4 import jieba
5 import string
6 import pickle
7 from collections import Counter
8 from collections import defaultdict
9
10 # 增加字段大小限制, 设置为更大的值 (比如 10MB)
11 csv.field_size_limit(10000000)
12
13 # 读取倒排索引
14 def load_index(filename="Text_Index.pkl"):
15     print("加载倒排索引...")
16     with open(filename, 'rb') as f:
17         inverted_index, doc_tfs, idf = pickle.load(f)
18     print("倒排索引加载完成。")
19     return inverted_index, doc_tfs, idf
20
21 # 打开并读取 Html_Content.csv 文件
22 csv_filename = "Html_Content.csv"
23
24 # 读取文件中的内容
25 with open(csv_filename, mode='r', encoding='utf-8') as file:
26     csv_reader = csv.reader(file)
27     # 跳过表头 (如果有的话)
28     next(csv_reader, None)
29     # 将所有内容读入列表
30     html_data = list(csv_reader)
31
32 # 加载倒排索引
33 inverted_index, doc_tfs, idf = load_index()
```

2.4.2 代码功能分析

- 首先, 导入了多个必要的库, 包括 **os** (操作系统相关功能)、**csv** (处理 CSV 文件)、**math** (数学函数)、**jieba** (中文分词)、**string** (字符串处理)、**pickle** (序列化与反序列化工具), 以及 **Counter** 和 **defaultdict** (用于计数和字典操作的工具)。
- 通过调用 **csv.field_size_limit(10000000)**, 设置了 CSV 文件的最大字段大小限制为 10MB。此操作防止读取过大的 CSV 文件时发生异常。
- **load_index** 函数的作用是从指定的文件 (默认为 "Text_Index.pkl") 加载倒排索引。该函数使用 **pickle.load** 从文件中反序列化数据, 并将倒排索引、文档词频 (**doc_tfs**)

以及逆文档频率 (idf) 一并加载。加载完毕后, 打印提示信息, 并返回这些数据。

- 接下来, 代码打开并读取名为 **Html_Content.csv** 的 CSV 文件。使用 **csv.reader** 读取文件内容, 并跳过文件的表头 (如果存在)。文件的所有内容被读入到 **html_data** 列表中, 供后续处理使用。
- 最后, 调用 **load_index** 函数加载倒排索引, 并将返回的倒排索引数据赋值给 **inverted_index**、**doc_tfs** 和 **idf**。

2.5 实现链接分析

2.5.1 操作实现代码

```
1 from urllib.parse import urlparse
2
3 # 中文分词
4 def tokenize(text):
5     return list(jieba.cut(text))
6
7 # 计算PageRank的函数 (采用简化的PageRank算法)
8 def compute_pagerank(html_folder, top_docs, d=0.85, max_iter=100, tol=1e-6):
9     # 构建网页之间的链接关系
10    link_graph = defaultdict(list)
11    doc_urls = []
12    for doc_id, score in top_docs:
13        # 获取网页的URL和锚文本
14        webpage_title = html_data[doc_id][0] # 第一列是 标题
15        webpage_url = html_data[doc_id][1]   # 第二列是 URL
16        doc_urls.append(webpage_url)
17        webpage_anchors = html_data[doc_id][2] # 第三列是 锚文本
18
19    # 构建本地文件路径
20    local_file_path = os.path.join(html_folder, f'{webpage_title}.html')
21
22    # 从本地HTML文件中解析链接
23    if os.path.exists(local_file_path):
24        with open(local_file_path, 'r', encoding='utf-8') as file:
25            content = file.read()
26            links = extract_links_from_html(content) # 提取HTML中的链接
27            for link in links:
28                # 如果该链接在我们关注的网页列表中, 则记录下来
29                if link in doc_urls:
30                    link_graph[webpage_url].append(link)
31
32    # 初始化PageRank
33    num_docs = len(doc_urls)
34    pagerank = {url: 1 / num_docs for url in doc_urls} # 每个页面的初始PageRank值为 1/N
35    for _ in range(max_iter):
36        new_pagerank = {}
37        for url in doc_urls:
```

```
38     inbound_links = [key for key, links in link_graph.items() if url in links]
39     rank_sum = sum(pagerank[link] / len(link_graph[link]) for link in
40                   inbound_links)
41     new_pagerank[url] = (1 - d) / num_docs + d * rank_sum
42
43     # 检查是否收敛
44     if all(abs(new_pagerank[url] - pagerank[url]) < tol for url in doc_urls):
45         break
46
47     pagerank = new_pagerank
48
49     return pagerank
```

2.5.2 代码功能分析

- **tokenize** 函数用于将输入的中文文本进行分词处理，利用 **jieba.cut** 方法将文本分割为词语并返回一个词语列表。该函数适用于中文文本的词语提取。
- **compute_pagerank** 函数实现了一个简化版的 PageRank 算法，功能是根据网页之间的链接关系计算每个网页的 PageRank 值。其主要步骤如下：
 - 构建网页之间的链接关系图 **link_graph**，并初始化一个空的网页 URL 列表 **doc_urls**。
 - 遍历传入的 **top_docs** 列表，该列表包含网页的 ID 和其相关的分数。对于每个网页，从 **html_data** 中提取标题、URL 和锚文本信息，并将网页 URL 存入 **doc_urls**。
 - 构建本地 HTML 文件路径 **local_file_path**，并从中提取网页的超链接。若该链接存在于 **doc_urls** 中，则在 **link_graph** 中记录该链接。
- 在初始化 PageRank 时，每个网页的初始值为 $1 / \text{num_docs}$ ，其中 **num_docs** 是网页总数。然后，通过迭代更新每个网页的 PageRank 值，直到达到最大迭代次数 **max_iter** 或收敛条件（即前后 PageRank 值变化小于 **tol**）。
- 在每次迭代中，通过计算每个网页的入链（**inbound_links**）的 PageRank 贡献值来更新其新的 PageRank 值。每个页面的更新公式为：

$$\text{new_pagerank}[url] = \frac{(1 - d)}{\text{num_docs}} + d \times \sum_{\text{link} \in \text{inbound_links}} \frac{\text{pagerank}[\text{link}]}{\text{len}(\text{link_graph}[\text{link}])}$$

其中 **d** 是阻尼系数，默认值为 0.85，表示链接传递 PageRank 的比例。

- 在每次迭代后，检查 PageRank 值是否收敛，即所有网页的 PageRank 值变化是否小于指定的容忍度 **tol**。若收敛，则提前终止迭代。

- 最终，返回计算得到的每个网页的 PageRank 值字典 **pagerank**。

2.6 向量空间模型

2.6.1 操作实现代码

```
1 # 从HTML中提取所有的链接（简化版本，仅提取href链接）
2 def extract_links_from_html(content):
3     from bs4 import BeautifulSoup
4     soup = BeautifulSoup(content, 'html.parser')
5     links = soup.find_all('a', href=True)
6     return [link['href'] for link in links]
7
8 # 计算查询的TF-IDF向量
9 def compute_query_tfidf(query, idf):
10     query_tokens = tokenize(query)
11     query_tf = defaultdict(int)
12
13     # 计算TF
14     for word in query_tokens:
15         query_tf[word] += 1
16     total_words = len(query_tokens)
17     for word in query_tf:
18         query_tf[word] /= total_words
19
20     # 计算TF-IDF
21     query_tfidf = {}
22     for word, tf in query_tf.items():
23         query_tfidf[word] = tf * idf.get(word, 0) # 默认IDF为0
24
25     return query_tfidf
26
27 # 计算文档与查询的相关度（余弦相似度）
28 def compute_cosine_similarity(query_tfidf, doc_tfidf):
29     dot_product = sum(query_tfidf.get(word, 0) * doc_tfidf.get(word, 0) for word in
30                        query_tfidf)
31     query_norm = math.sqrt(sum(val**2 for val in query_tfidf.values()))
32     doc_norm = math.sqrt(sum(val**2 for val in doc_tfidf.values()))
33     if query_norm * doc_norm == 0:
34         return 0
35     return dot_product / (query_norm * doc_norm)
36
37 # 执行搜索
38 def search(query, inverted_index, doc_tfs, idf, top_n=10):
39
40     # 计算查询的TF-IDF向量
41     query_tfidf = compute_query_tfidf(query, idf)
42
43     # 计算每个文档与查询的相关度
44     scores = []
45     for doc_id, doc_tfidf in enumerate(doc_tfs):
46         score = compute_cosine_similarity(query_tfidf, doc_tfidf)
47         scores.append((doc_id, score))
```

```
47
48 # 按相关度排序并返回前top_n个文档
49 scores.sort(key=lambda x: x[1], reverse=True)
50 top_docs = scores[:100]
51
52 # 计算PageRank
53 html_folder = './HtmlFile/'
54 pagerank = compute_pagerank(html_folder, top_docs)
55
56 # 综合文档评分与PageRank
57 combined_scores = []
58 for doc_id, score in top_docs:
59     webpage_url = html_data[doc_id][1] # 获取文档URL
60     pagerank_score = pagerank.get(webpage_url, 0) # 获取该网页的PageRank
61     combined_score = score + pagerank_score # 综合评分
62     combined_scores.append((doc_id, combined_score))
63
64 # 按综合评分重新排序并返回前top_n个文档
65 combined_scores.sort(key=lambda x: x[1], reverse=True)
66 top_combined_docs = combined_scores[:top_n]
67
68 return top_combined_docs
```

2.6.2 代码功能分析

- **extract_links_from_html** 函数的作用是从 HTML 文档中提取所有的超链接。它使用了 **BeautifulSoup** 库来解析 HTML 内容,并通过 `soup.find_all('a', href=True)` 方法查找所有带有 `href` 属性的 `a` 标签。最终返回一个包含所有链接的列表。
- **compute_query_tfidf** 函数用于计算查询的 TF-IDF 向量。首先,通过 **tokenize** 函数将查询文本分词,然后计算每个词的词频 **query_tf**,并根据查询总词数对每个词的词频进行归一化。接着,利用词频与逆文档频率 **idf** 来计算每个词的 TF-IDF 值,并返回最终的查询 TF-IDF 向量。
- **compute_cosine_similarity** 函数计算查询向量与文档向量之间的余弦相似度。通过对查询向量与文档向量的点积计算,结合向量的范数来得出余弦相似度。如果任一向量的范数为 0,则返回 0,否则返回余弦相似度值。
- **search** 函数实现了一个基于 TF-IDF 和 PageRank 的搜索引擎功能。首先,计算查询的 TF-IDF 向量 **query_tfidf**,然后计算每个文档与查询之间的余弦相似度得分。根据得分排序后,选择前 **top_n** 个文档。接着,通过调用 **compute_pagerank** 函数计算 PageRank 值,并将 PageRank 值与余弦相似度得分结合,得到综合评分。最终,根据综合评分排序并返回排名前 **top_n** 的文档。

2.7 用户系统实现

2.7.1 操作实现代码

```
1  # 读取用户信息
2  def load_user_info(file_path):
3      users = {}
4      if os.path.exists(file_path):
5          with open(file_path, mode='r', newline='', encoding='utf-8') as file:
6              reader = csv.reader(file)
7              for row in reader:
8                  account, password, nickname, preferences = row
9                  users[account] = {'password': password, 'nickname': nickname, '
                                preferences': preferences}
10     return users
11
12 # 保存用户信息
13 def save_user_info(file_path, users):
14     with open(file_path, mode='w', newline='', encoding='utf-8') as file:
15         writer = csv.writer(file)
16         for account, info in users.items():
17             writer.writerow([account, info['password'], info['nickname'], info['
                                preferences']])
18
19 # 用户注册
20 def register(users):
21     print("用户注册")
22     account = input("请输入账号: ")
23     if account in users:
24         print("账户已存在, 请选择其他账户名.")
25         return None
26     password = input("请输入密码: ")
27     nickname = input("请输入昵称: ")
28     preferences = input("请输入您的偏好设置 (例如: '学术, 科技'): ")
29     users[account] = {'password': password, 'nickname': nickname, 'preferences':
                                preferences}
30     print(f"注册成功! 欢迎, {nickname}")
31     return account
32
33 # 用户登录
34 def login(users):
35     print("用户登录")
36     account = input("请输入账号: ")
37     if account not in users:
38         print("账号不存在, 请先注册.")
39         return None
40     password = input("请输入密码: ")
41     if users[account]['password'] == password:
42         print(f"登录成功! 欢迎, {users[account]['nickname']}")
43         return account, users[account]['preferences']
44     else:
45         print("密码错误, 请重新尝试.")
46     return None
```


2.7.2 代码功能分析

以下是对代码的功能分析：

- **load_user_info(file_path)**: 该函数负责从指定路径的文件中读取用户信息。若文件存在，函数会逐行读取文件内容，并解析每行的数据（包括账户、密码、昵称和偏好设置），然后将这些信息存储在一个字典 **users** 中。每个账户是字典 **users** 的键，而每个账户的相关信息（密码、昵称、偏好设置）是字典的值。
- **save_user_info(file_path, users)**: 该函数负责将用户信息（存储在字典 **users** 中）保存到指定路径的文件中。每个账户的信息会以 CSV 格式写入文件，其中每一行包括账户、密码、昵称和偏好设置。
- **register(users)**: 该函数实现用户注册功能。用户输入账号、密码、昵称和偏好设置。如果用户输入的账户名已经存在，函数会提示用户重新选择账户名。否则，注册信息会被添加到 **users** 字典中，并提示用户注册成功。
- **login(users)**: 该函数实现用户登录功能。用户输入账号和密码，系统会检查该账户是否存在，以及密码是否正确。如果账号不存在，系统提示用户先注册；如果密码错误，系统提示用户重新尝试；如果登录成功，系统会显示欢迎信息，并返回用户的账户名和偏好设置。

2.8 其余部分实现

2.8.1 操作实现代码

```
1 # 定义循环次数计数器
2 search_count = 0
3
4 # 初始化查询日志
5 query_log = []
6
7 # html 文件存放在当前目录下的 HtmlFile 文件夹
8 html_folder = './HtmlFile/'
9
10 # 确保 HtmlFile 文件夹存在
11 if not os.path.exists(html_folder):
12     print("HtmlFile 文件夹不存在，请检查文件路径。")
13     exit()
14
15 # 定义一个用于过滤标点符号的函数
16 def is_valid_word(word):
17     invalid_symbols = [' ', '%', '%o', ', ', '']
18     invalid_words = ['与', '本', '的', '由', '多', 'l', '新']
19
20     if any(symbol in word for symbol in invalid_symbols) or word in invalid_words:
```

```
21         return False
22
23     return all(char not in string.punctuation for char in word)
24
25 # 分词并统计双词
26 def extract_bigrams(text):
27     words = list(jieba.cut(text)) # 切割文本
28     bigrams = []
29     for i in range(len(words) - 1):
30         bigram = f"{words[i]} {words[i+1]}"
31         if is_valid_word(words[i]) and is_valid_word(words[i+1]) and words[i] != words[
            i+1]:
32             bigrams.append(bigram)
33     return bigrams
34
35 # 主程序
36 def main():
37     # 加载用户信息
38     user_info_file = 'User_Information.csv'
39     users = load_user_info(user_info_file)
40
41     # 注册或登录
42     current_user = None
43     preferences = None
44     while current_user is None:
45         action = input("请选择操作：1. 注册 2. 登录 (输入 'exit' 退出): ")
46         if action == '1':
47             current_user = register(users)
48             preferences = users[current_user]['preferences'] if current_user else None
49         elif action == '2':
50             result = login(users)
51             if result:
52                 current_user, preferences = result
53         elif action.lower() == 'exit':
54             print("退出程序")
55             return
56         else:
57             print("无效选项，请选择 1 或 2。")
58
59     # 保存用户信息
60     save_user_info(user_info_file, users)
61
62     # 初始化查询日志
63     query_log.clear()
64
65     # 循环进行查询
66     global search_count
67     while True:
68         # 根据循环次数决定搜索内容
69         if search_count == 0:
70             query = "南开大学ESI学科发展报告pdf" # 文档查询测试
71         elif search_count == 1:
72             query = "南开大学新校区" # 短语查询测试
73         elif search_count == 2:
74             query = "新冠肺炎防控*漫画" # 通配查询测试
75         elif search_count == 3:
```

```
76     #     query = "exit"
77     # else:
78     #     query = input("请输入查询词 (输入 'exit' 退出): ")
79     query = input("请输入查询词 (输入 'exit' 退出): ")
80
81     # 如果用户输入 "exit", 退出循环
82     if query.lower() == 'exit':
83         print("\n退出程序。")
84         break
85
86     # 输出查询内容
87     print(f"\n正在搜索: {query}")
88
89     # 记录查询日志
90     query_log.append(query)
91
92     # 如果用户已登录, 则将用户偏好附加到查询词
93     if preferences:
94         query = f"{query} {preferences}"
95
96     # 执行搜索 (假设 search 是一个已定义的搜索函数)
97     top_docs = search(query, inverted_index, doc_tfs, idf, top_n=10)
98
99     # 输出搜索结果
100    print("搜索结果: ")
101
102    all_bigrams = [] # 用于收集所有搜索结果中的双词
103
104    for doc_id, score in top_docs:
105        webpage_title = html_data[doc_id][0] # 假设第一列是 标题
106        webpage_url = html_data[doc_id][1] # 假设第二列是 URL
107        webpage_anchors = html_data[doc_id][2] # 假设第三列是 锚文本
108        webpage_body = html_data[doc_id][3] # 假设第四列是 正文
109
110        all_bigrams.extend(extract_bigrams(webpage_anchors))
111        all_bigrams.extend(extract_bigrams(webpage_body))
112
113        # 构建本地文件路径 (假设网页标题对应文件名)
114        local_file_path = os.path.join(html_folder, f"{webpage_title}.html")
115
116        if os.path.exists(local_file_path):
117            local_file_path = local_file_path.replace(' ', '%20')
118            local_file_url = f"file:/// {os.path.abspath(local_file_path)}"
119            print(f"网页标题: {webpage_title}, 网页链接: {webpage_url}\n本地文件链
120                接: {local_file_url}")
121        else:
122            print(f"网页标题: {webpage_title}, 网页链接: {webpage_url}\n本地文件不
123                存在")
124
125    # 输出查询日志
126    print("\n查询日志: ")
127    for i, log in enumerate(query_log, 1):
128        # if log.endswith(preferences):
129        #     log = log[:-len(preferences)]
130        print(f"{i}. {log}")
```

```
130     # 输出当前用户的昵称和偏好
131     if current_user:
132         print(f"\n当前用户: {users[current_user]['nickname']}")
133         print(f"用户偏好: {preferences}")
134
135     # 统计双词
136     bigram_counts = Counter(all_bigrams)
137     common_bigrams = bigram_counts.most_common(10)
138     print("\n推荐搜索: ")
139
140     # 输出推荐搜索短语
141     for i in range(0, len(common_bigrams), 10):
142         line = ", ".join([bigram for bigram, _ in common_bigrams[i:i+10]])
143         print(line)
144
145     search_count += 1
```

2.8.2 代码功能分析

以下是对代码的功能分析：

- **search_count**: 这是一个全局变量，用于跟踪查询次数。在主程序中，**search_count** 用于控制查询的执行顺序，但实际代码中这一部分已被注释掉。
- **query_log**: 该变量是一个列表，用于存储用户输入的查询词。每次用户进行查询时，查询词都会被添加到 **query_log** 中，以便后续查看和记录。
- **html_folder**: 该变量指定了存放 HTML 文件的目录，即当前目录下的 **HtmlFile** 文件夹。如果文件夹不存在，程序会输出提示信息并退出。
- **is_valid_word(word)**: 该函数用于过滤标点符号和一些常见的无意义词汇（如“与”、“的”、“由”等）。它通过检查单词是否包含无效符号，或者是否在无效词汇列表中，来决定该单词是否有效。
- **extract_bigrams(text)**: 该函数用于从输入文本中提取双词（bigram）。首先，通过 **jieba.cut** 进行分词，然后过滤掉无效词汇和标点符号。最后，返回一个包含所有有效双词的列表。
- **main()**: 该函数是主程序，负责用户交互、查询处理和搜索结果展示。主要功能包括：
 - 加载用户信息并提供注册或登录选项。
 - 如果用户已经登录，允许用户进行查询。每次查询时，查询词会记录在 **query_log** 中，并与用户的偏好设置（**preferences**）结合。

- 执行查询后，输出搜索结果，并通过 **extract_bigrams** 提取每个文档中的双词。
 - 如果文档存在本地文件，程序会输出本地文件链接；否则，输出文档的网页链接。
 - 显示查询日志，输出用户昵称和偏好设置，并统计所有搜索结果中的双词频率。
 - 输出最常见的 10 个双词，作为推荐搜索短语。
- **search_count** 的作用：这个变量用于控制程序中查询的次数。虽然在代码中它的具体作用被注释掉，但原本它可以通过不同的条件决定每次查询的内容。查询次数达到一定值后，可以通过输入词进行自由查询。
 - **Counter(all_bigrams)**: 在查询结果中，**all_bigrams** 保存了所有文档中提取的双词。通过 **Counter** 类，可以统计各个双词的出现频率，并输出最常见的双词作为推荐搜索短语。
 - **搜索结果展示与文件路径处理**: 对于每一个搜索结果，程序展示了网页标题、网页链接，并检查本地是否有该网页的 HTML 文件。如果文件存在，则显示本地文件链接；否则，显示网页的 URL。
 - **用户信息保存**: 在程序运行结束时，用户信息会被保存回 CSV 文件中，确保用户的注册信息得以持久化。

三、 实验结果

3.1 文档查询截图

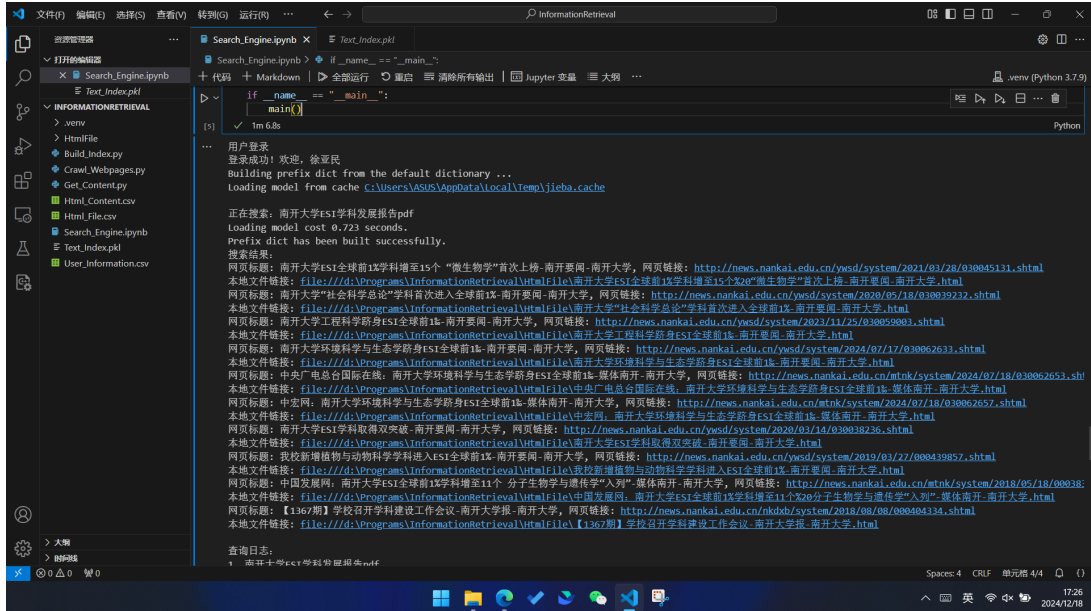


图 4: 文档查询截图

3.2 短语查询截图

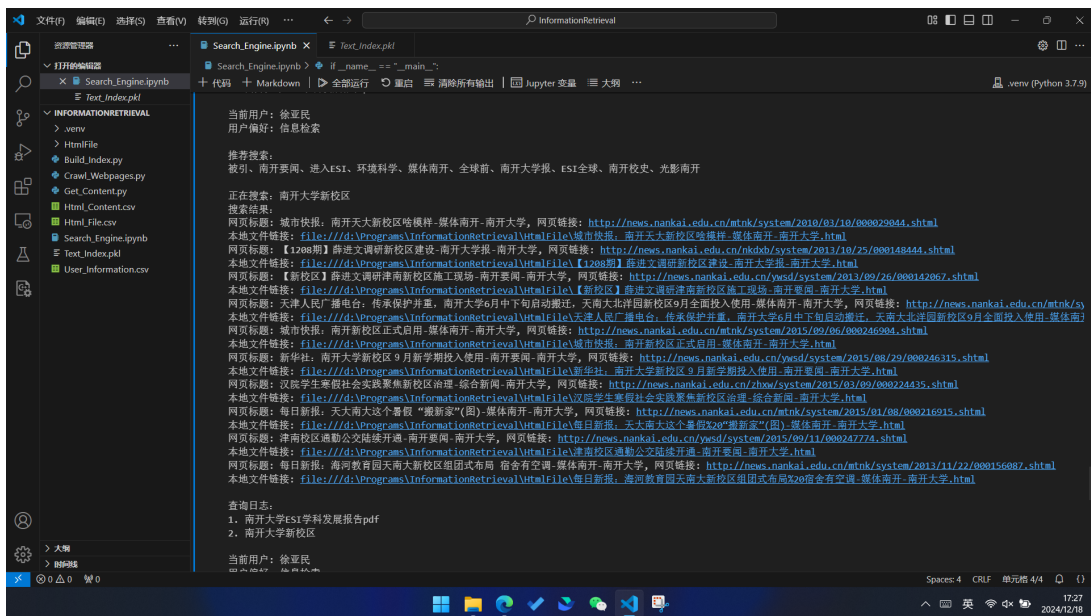


图 5: 短语查询截图

3.3 通配查询截图

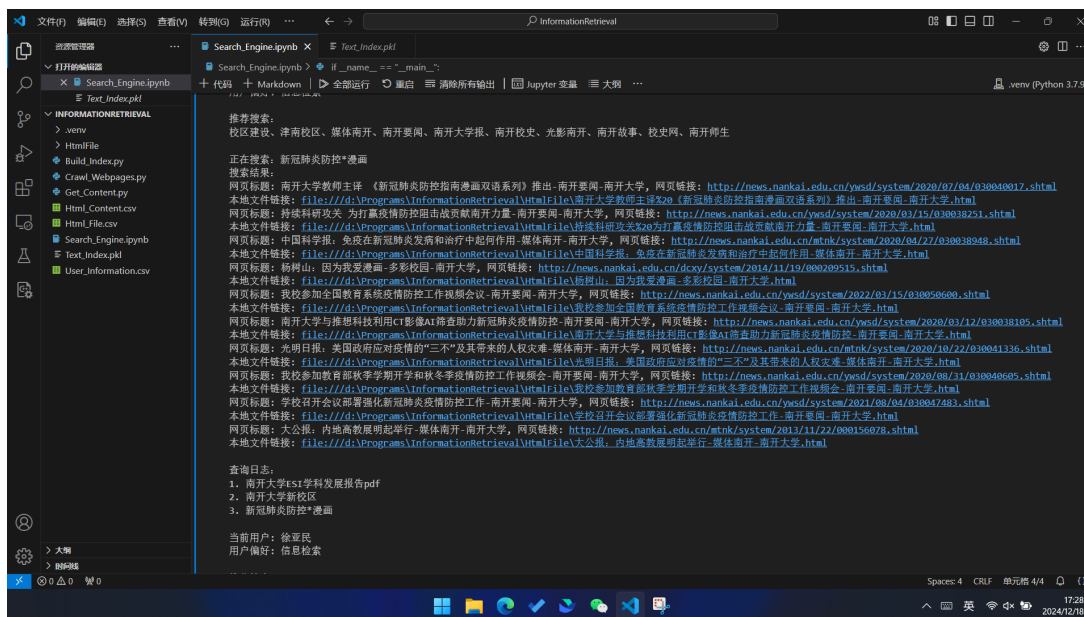


图 6: 通配查询截图

3.4 网页快照截图



图 7: 网页快照截图

3.5 其余功能截图

下方截图体现了：查询日志、个性化查询、个性化推荐、终端交互。

具体演示细节请您查看附件视频。

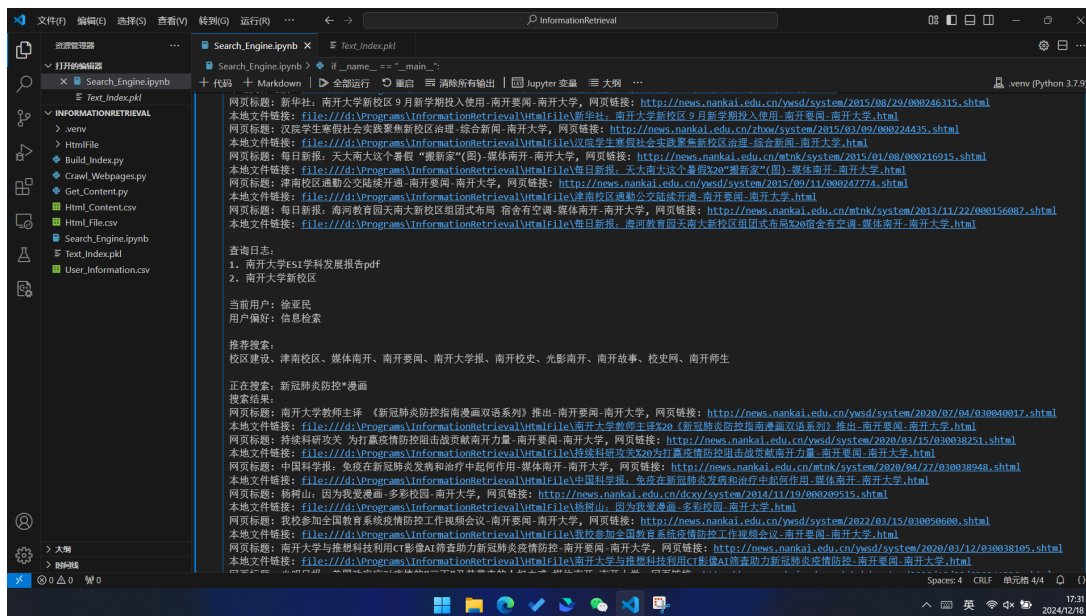


图 8: 其余功能截图

四、 实验总结

本次实验的目标是构建一个基于南开校内资源的 Web 搜索引擎，涵盖了网页抓取、文本索引、链接分析、查询服务、个性化查询、Web 界面以及个性化推荐等多个方面。在本次实验中，我的实现重点集中在利用现有的工具和技术（如 ElasticSearch）高效地处理和索引大量的网页数据，提供丰富的查询功能，并通过个性化推荐提升用户的体验。以下是我在实现过程中的一些经验总结：

4.1 网页抓取

在抓取网页时，我遵循了爬虫协议并设置了合理的爬取间隔，确保不会对南开校园网站造成过大的负担。通过利用 Python 中的爬虫工具（如 Scrapy 和 BeautifulSoup），我成功地抓取了超过 10 万个网页数据。为了确保抓取的网页内容准确和完整，我特别注意了抓取时的编码格式和数据清洗问题，确保了最终数据的高质量。

4.2 文本索引与链接分析

为了支持高效的查询服务，我使用了 Elasticsearch 来进行网页内容的索引，包含了网页标题、URL、锚文本等多个字段的索引。通过配置合理的分词器和权重模型，我能够确保搜索结果的相关性和准确性。

在链接分析方面，我实现了 PageRank 算法来评估网页的权重，结合网页间的链接关系进行排名。这一部分显著提升了搜索结果的排序质量，尤其在网页内容相似时，PageRank 的效果尤为突出。

4.3 查询服务

为了满足用户的多样化查询需求，我实现了六种不同的查询模式：站内查询、文档查询、短语查询、通配查询、查询日志和网页快照。这些查询方式通过向量空间模型和链接分析相结合的方式优化搜索结果排序。

- **站内查询**：用户可以通过输入简单的关键字来查找南开校内的网页内容。
- **文档查询**：实现了对附件类型文件（如 DOC、PDF、XLS）进行索引和搜索，确保用户可以轻松查找到相关文件。
- **短语查询**：支持多词查询和语义理解，在处理“南开 1 大学 2”和“南开 1 是一所综合性大学 2”时能够正确区分不同的查询意图。
- **通配查询**：用户可以通过使用通配符（如“温”和“计?”）来进行模糊查询，提供了灵活的查询方式。
- **查询日志**：记录用户的查询历史，便于后续分析并为个性化推荐服务提供数据支持。
- **网页快照**：实现了对网页内容的备份功能，当用户点击链接时可以查看当时抓取的网页快照。

4.4 个性化查询与推荐

个性化查询是本次实验的一个重要组成部分。在查询服务中，我通过为用户提供注册/登录系统来收集用户的兴趣和历史数据，基于这些数据对搜索结果进行排序优化。此外，我还实现了个性化推荐系统，通过分析用户行为和查询日志，推荐相关内容，以提升用户体验。

4.5 Web 界面与交互

虽然本次实验的重点并不在于界面的美观性，但我实现了一个简洁的终端交互界面。用户可以通过命令行输入查询内容，系统将返回相应的搜索结果和推荐内容。通过终端交互，我可以清晰地展示搜索引擎的功能，并为用户提供简单而有效的查询体验。

4.6 挑战与改进空间

在实现过程中，我遇到了一些挑战，主要集中在以下几个方面：

- **数据清洗**：抓取的网页数据格式不统一，需要进行大量的数据清理和预处理工作，确保文本的有效性和一致性。
- **查询性能优化**：随着数据量的增加，查询的响应速度出现了瓶颈。未来可以通过增加缓存机制、优化查询算法等方式进一步提升性能。
- **个性化推荐算法**：虽然我实现了基于查询日志的简单推荐算法，但在处理用户兴趣建模时，仍有进一步改进的空间，特别是在深度学习和自然语言处理技术的应用方面。

总的来说，本次实验不仅加深了我对搜索引擎原理的理解，也让我学会了如何结合多种技术实现一个完整的 Web 搜索引擎。未来，我会继续优化系统性能，并探索更多基于用户行为的推荐算法，使得搜索引擎更加智能和高效。