实验报告

课程名称 信 息 检 索

专业年级

姓　　名

学　　号

协 作 者 无

实验学期 2020-2021 学年 第二 学期

课堂时数 32 课外时数 32

填写时间 2021 年 6 月 19 日

实验介绍
【实验名称】：英文文本检索系统

【实验目的】：

开发一款针对英文文本的信息检索系统，可以实现建立索引表、布尔查询、通配符查询、短语查询等功能，并通过开发过程达到以下目的：

（1）复习本学期所学信息检索知识；

（2）掌握基本的信息检索方法，了解检索系统的搭建；

（3）具备实现、维护与优化信息检索系统的能力。

目前实现的功能有：

（1）自动获取某英文小说网站的文本作为数据源；

（2）建立查询表；

（3）计算指定词的 TF-IDF 值；

（4）进行布尔查询；

（5）进行通配符查询；

（6）进行短语查询。

所有功能都可以通过—hit 参数限制输出的结果数量。

【实验环境】：

(1)处理器：

Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz　2.40 GHz

(2)操作系统环境：

Windows 10 家庭中文版 x64 21H1 19043.1052

(3)编程语言：

Python 3.8

(4)IDE 及包管理器：

JetBrains PyCharm 2020.1 x64， anaconda 3 for Windows（conda 4.9.0）

(6)使用的第三方库：

见附件 requirements.txt

【参考文献】：

[1]. [美]克里斯托夫·曼宁，[美]普拉巴卡尔·拉格万，[德]欣里希·舒策 著.王斌，李鹏 译.信息检索导论（修订版）.人民邮电出版社,2019.7.

实验内容

【实验方案设计】：

本部分将围绕以下 8 个模块，就原理和实现层面分别予以介绍：用户交互的实现、数据获取、查询表的建立、布尔查询、TF-IDF 值的计算、通配符查询、短语查询、结果数目更改。

1、用户交互的实现：

考虑到信息检索更像是一个具有一些功能的集成系统而非寥寥数个特定的功能，因此决定在开发时采用交互式设计。

原理上，考虑采用 python 中的 cmd 模块。cmd 模块是 python 中包含的一个公共模块，是一个用于交互式 shell 和其它命令解释器等的基类。因此可以考虑基于 cmd 模块自定义一个子类，实现我们自己的交互式 shell。

cmd 模块的执行流程十分简单。输入行包括两个部分：命令和参数，使用命令行解释器循环读取输入的所有行并对其进行解析，然后把解析后的输入行交给命令处理器来处理。通过继承和子类方法重载父类方法的特性，命令行处理器就可以找到适合处理该命令的子类方法。

具体实现上，首先构建一个 IRcmder 类，让它继承自 python 中的 cmd 模块：

```
if __name__ == '__main__':
    info = "\n\nThis is a simple Information Retrival System.\nCopyright 2021 " \
           "@CandyMonster37: https://github.com/CandyMonster37\n" + \
           "A course final project for Information Retrival, and you can find the latest vertion of the codes " \
           "here: \n   https://github.com/CandyMonster37/InformationRetrival.git \n\n\n"

    print(info)

    IRcmder.prompt = 'IR > '
    IRcmder().cmdloop()
```

```
class IRcmder(cmd.Cmd):
    intro = "Welcome to the Information Retrival System.\n".center(100, ' ')
    intro += "\n\nThis is a simple Information Retrival System.\n"
    intro += "You can use some commands to do some work related to the information retrieval.\n"
    intro += "Only supports English.\n"
    intro += "Shell commands are defined internally.  \n\n"
    intro += "Type \'help\' or \'?\' to list all available commands.\n"
    intro += "Type \'help cmd\' to see more details about the command \'cmd\'.\n"
    intro += "Or type \'exit\' to exit this system.\n\n"

    def __init__(self):
        super(IRcmder, self).__init__()
        self.k = 10
```

随后再定义一些解析方式和处理逻辑：

```
    def do_exit(self, args):
        try:
            print('\nThank you for using. Goodbye.\n')
            sys.exit()
        except Exception as e:
            print(e)

    def emptyline(self):
        pass

    def default(self, line):
        print('Unrecognized command.\nNo such symbol : {0}'.format(line))

    def help_build_table(self):
        cmd_info = 'command: \tbuild_table'.center(80, ' ')
```

继承自 cmd 的 IRcmder 就可以提供一个类似命令行的环境，这样用户在使用时就可以达到运行一个脚本、使用多个功能的目的。

2、数据的获取：

数据可以由用户自己提供，也可以由本系统自行抓取。如果由用户自行提供，则用户须将所有的 txt 文本放置于某个文件夹下，并在之后的操作中将文件夹路径作为参数传入系统。若由本系统自行抓取，则需输入 get_data 命令，随后系统会从某小说网站上爬取一定数量的英文短篇小说作为数据源。该网站的架构为静态页面，比较好爬取。

关于爬虫部分，首先会对用户传进来的参数进行解析，以获取用户想要的文章数、停等时间、存放目录等信息：

```
    def do_get_data(self, args):
        # get_data ./data --numbers=10 --wait=0.5
        k = 10
        un_mattched = args.split(' ')
        hit_arg_rule = r'(?<=numbers=)[\w]*'
        for item in un_mattched:
            res = re.search(hit_arg_rule, item)
            if res:
                un_mattched.remove(item)
                k_rule = r'(?<=numbers=)[\d]*'
                k = re.search(k_rule, item).group()
                break
        args = ' '.join(un_mattched)
        try:
            tar_k = int(k)
            k = tar_k
        except Exception as e:
            print(e)
        tar_seconds = None
        un_mattched = args.split(' ')
        wait_arg_rule = r'(?<=wait=)[\w]*'
        for item in un_mattched:
            res = re.search(wait_arg_rule, item)
            if res:
                un_mattched.remove(item)
                wait_rule = r'(?<=wait=)[\d]*'
                seconds = re.search(wait_rule, item).group()
                try:
                    tar_seconds = int(seconds)
                except Exception as e:
                    print(e)
                break
        args = ' '.join(un_mattched)
        dirr = args.strip(' ')
        if not os.path.exists(dirr):
            os.mkdir(dirr)
        bug = Spider(limit=k, save_dir=dirr)
        bug.get_novels(wait=tar_seconds)
        bug.get_chapter(wait=tar_seconds)
```

该部分通过两个正后发断言正则表达式构成，通过两个正则匹配以获取用户想要存放文章的文件集以及停等时间参数。随后通过调用一个 Spider 类实现爬取流程。Spider 其实主要

是对网页的请求、解析以及对数据的保存，通过 request 库和 beautiful soup 库实现。

```python
def get_html(url, show=True):
    headers = {'Connection': 'close', 'User-Agent': random.choice(ua_list)}

    try:
        res = requests.get(url, headers=headers)
        if show:
            print('url:', url)
            print('get response successfully! ', time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))
        # print(res.status_code)
        res.encoding = res.apparent_encoding
        html = res.text
        return html
    except requests.HTTPError as e:
        print('http error: status_code', e)
        return ""
    except Exception as e:
        print('other error:')
        print(e)
        return ""
```

爬虫在爬取时会随机选择一个 UA 作为浏览器头，然后对目标链接进行访问。访问成功后会根据目的进行解析，比如解析章节列表或解析文本内容：

```python
def get_novels(self, wait=None):
    self.t_start = time.time()
    piece = 10
    pages = math.ceil(self.limit / piece)
    for i in range(pages):
        if len(self.novel_url_dic) >= self.limit:
            break
        rela_add = '/duanpian/list_31_{0}.html'.format(i + 1)
        tar = self.base_url + rela_add
        html_page = get_html(url=tar, show=True)
        if not html_page:
            continue
        soup = bs(html_page, 'html.parser')
        novels_url_path = 'body > div.zhongyb > div.zhongz > div > div > div.all0@1xp1 > div > div.text > h6 > a'
        novels_url = soup.select(novels_url_path)
        for item in novels_url:
            novel = self.base_url + item.get('href')
            novel_name = item.text
            if novel not in self.novel_url_dic:
                self.novel_url_dic[novel_name] = novel
            if len(self.novel_url_dic) >= self.limit:
                break
        if wait:
            time.sleep(wait)
```

爬虫部分的处理工作大同小异，故不再赘述，仍以信息检索相关工作为重点进行介绍。

3、查询表的建立

在使用整个系统进行信息检索相关工作之前需要先构建查询表，该部分通过命令 build_table 实现。构建完查询表之后才能进行布尔检索模型、通配符查询模型、短语查询模型等的工作。该部分的工作主要由四部分构成：预处理、倒排表构建、倒排表压缩、轮排索引构建。

```python
    # 构建倒排表
    def do_build_table(self, args):
        try:
            self.object = process(args)
            self.object.indextable.index_compression()
            self.object.indextable.create_Permuterm_index()

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

3.1 预处理

预处理主要包括切词、词条化、词归一化等操作：

```python
def process(args):
    pre_mattched = args.split(' ')  # do re
    rule = r'(?<=language=)[\w]*'
    lan = ''
    for item in pre_mattched:
        res = re.search(rule, item)
        if res:
            lan = res.group()
            pre_mattched.remove(item)
            break
    if not lan:
        lan = 'en'
    dir_name = pre_mattched[0]

    print('Begin loading and build index.')
    objects = StaticObjects()
    objects.documents, objects.doc_lists = readfiles(dir_name)
    objects.document_words = tokenize(objects.documents, language=lan)
    objects.indextable = IndexTable(objects.document_words)

    for words in objects.document_words.items():
        for word in words[1]:
            objects.indextable.insert_pair(word, words[0])
        for i in range(len(words[1]) - 1):
            objects.indextable.insert_pair_2(words[1][i] + ' ' + words[1][i + 1], words[0])
    # print(objects.indextable.table)
    print('Finished loading and build index.\n')

    return objects
```

在对用户输入的文本语言进行解析后，process 通过 readfiles 方法读取文件：

```python
def readfiles(dir_name):
    documents = {}
    doc_lists = []
    for file_name in os.listdir(dir_name):
        try:
            tar = os.path.join(dir_name, file_name)
            with open(tar, 'r', encoding='utf8') as f:
                text = f.read()
                doc_lists.append(tar)
                documents[doc_lists.index(tar)] = text
        except Exception as e:
            print(e)
            print(file_name)
    return documents, doc_lists
```

该方法会将文件名映射为文档 ID，并读取文档内容。除了输出结果时会用到文档名之外，之后所有涉及到文档名的操作都会通过文档 ID 来替代，以达到节省内存和提高性能的目的。process 随后调用 tokenize 方法对其进行词条化处理：

```python
def tokenize(documents, language='en'):
    document_words = {}
    if language == 'en':
        for _doc in documents.items():
            doc = []
            tar_str = _doc[1]
            words = tar_str.replace('\n', ' ').split(' ')
            for item in words:
                if item:
                    word = re.search(r'[\w]*[\d]*', item).group()
                    # 只保留单词
                    doc.append(word)
            document_words[_doc[0]] = doc
        return document_words
    elif language == 'zh':
        for _doc in documents.items():
            doc = []
            tar_str = _doc[1]
            cutted = jieba.cut(tar_str, cut_all=False)
            for item in cutted:
                if item in punctuations:
                    continue
                elif item:
                    doc.append(item)
            document_words[_doc[0]] = doc
        return document_words
    else:
        print("no support language")
        raise
```

tokenize 会根据用户传进来的参数分别用对英文的方法和对中文的方法对文本进行词条化处理，其中针对中文主要用到 jieba 库，针对英文则通过简单的分割以及正则匹配实现切词。虽然这里有针对中文的解析，但实际上系统目前是不支持对中文的信息检索工作的，因为中文处理起来稍有些麻烦且被时间所迫，所以后续的工作中并不支持对中文的检索操作。

3.2 倒排表的构建

切词完成后会将切好的结果以{文档:词列表}的格式存储到一个字典对象中，随后 process 将其传入 IndexTable 类中初始化。

IndexTable 类是索引表的集合类，包括索引表的定义以及各种操作方法：

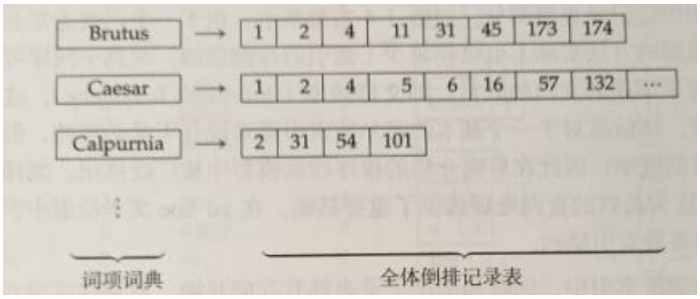
```
78  class IndexTable:
79      def __init__(self, document_words):
80          self.tep_table = {}
81          self.table = None
82          self.tep_table_2 = {}
83          self.table_2 = None
84          self.document_words = document_words
85          self.permuterm_index_table = False
86          self.length = 0
87          self.length_2 = 0
```

此时仅得到了一片片文档以及文档中的每一个词，并没有进行任何统计工作。之后 process 会将结果中的词频等数量信息进行统计，得到一个{词项:文档 ID 列表}的统计结果。在统计完成后 process 通过调用 IndexTable 的 insert_pair 和 insert_pair_2 的方法完成索引表的构建。insert_pair 和 insert_pair_2 的方法如下图所示：

```
89      def insert_pair(self, word, docID):
90          IDlist = self.tep_table.get(word, 'null')
91          if IDlist != 'null':
92              if IDlist[0].get(docID, 'null') != 'null':
93                  IDlist[0][docID] += 1
94              else:
95                  IDlist[0][docID] = 1
96                  IDlist[1] += 1
97          else:
98              self.tep_table[word] = [{docID: 1}, 1]
99              self.length += 1
100
101     def insert_pair_2(self, word, docID):
102         IDlist = self.tep_table_2.get(word, 'null')
103         if IDlist != 'null':
104             if IDlist.get(docID, 'null') != 'null':
105                 IDlist[docID] += 1
106             else:
107                 IDlist[docID] = 1
108         else:
109             self.tep_table_2[word] = {docID: 1}
110             self.length_2 += 1
```

这两个函数的逻辑是一样的，只是会根据不同的使用目的将结果存储到不同的对象中。这两个函数会将统计结果的词按字母序进行排序，然后将同一词项合并，最后将词项和文档 ID 分开的同时统计词频信息。最终将词项连通其文档频率一起存储在词典中，每个词项都

有一个指针指向自己的倒排记录表（此处借用 python 中的字典对象实现），倒排表存储了词项出现的文档列表及词项在对应文档中出现的次数。在所有的词项及文档都被遍历完后，一个全体倒排记录表也就构建完成了，其结构类似下图所示：



摘自《信息检索导论》第 1 章第 2 节

3.3 倒排表的压缩

考虑到当文档数目很大时，系统的性能可能有一定程度的下降，同时内存开销也会有一定程度的增大。因此，为了获得较好的查询性能，有必要对倒排表采用一定的压缩算法。这里采用《信息检索导论》第 5 章第 3 节第 1 小节所提到的 VB 编码对其进行压缩。

VB（Variable byte, 可变字节）编码利用整数个字节来对间距编码。字节的后 7 位是间距的有效编码区，而第 1 位是延续位（continuation bit）。如果该位为 1，则表明本字节是某个间距编码的最后一个字节，否则不是。要对一个可变字节编码进行解码，可以读入一段字节序列，其中前面的字节的延续位都为 0，而最后一个字节的延续位为 1。根据上述标识可以把每个字节的 7 位部分抽取出来并连接在一起形成编码。下表给出了一个采用 VB 编码的例子：

表5-4 VB编码。间距采用整数字节进行编码。每个字节中第一位为延续位，标识本次编码的结束(1)与否(0)

文档ID	824	829	215406
间距		5	214577
VB编码	00000110 10111000	10000101	00001101 00001100 10110001

摘自《信息检索导论》5.3.1 表 5-4

下图给出了 VB 编码和解码的伪代码：

```
VBENCODENUMBER(n)
1  bytes ← ⟨⟩
2  while true
3  do PREPEND(bytes, n mod 128)
4     if n < 128
5        then BREAK
6     n ← n div 128
7  bytes[LENGTH(bytes)] += 128
8  return bytes

VBENCODE(numbers)
1  bytestream ← ⟨⟩
2  for each n ∈ numbers
3  do bytes ← VBENCODENUMBER(n)
4     bytestream ← EXTEND(bytestream, bytes)
5  return bytestream

VBDECODE(bytestream)
1  numbers ← ⟨⟩
2  n ← 0
3  for i ← 1 to LENGTH(bytestream)
4  do if bytestream[i] < 128
5        then n ← 128 × n + bytestream[i]
6        else n ← 128 × n + (bytestream[i] − 128)
7             APPEND(numbers, n)
8             n ← 0
9  return numbers
```

摘自《信息检索导论》5.3.1 图 5-8

据此写出 VB 压缩编码的代码实现如下图所示：

```python
# -*- coding: utf-8 -*-
def vb_encode(doc_ids):
    byte_stream = []
    for num in doc_ids:
        bytes_split = []
        one_bytestream = ''
        while True:
            bytes_split.append((num % 128))
            if num < 128:
                break
            else:
                num = int(num / 128)
        bytes_split = bytes_split[::-1]
        for i in range(len(bytes_split)):
            if i == len(bytes_split) - 1:
                byte_stream.append(bytes([bytes_split[i] + 128]))
            else:
                byte_stream.append(bytes([bytes_split[i]]))
    return byte_stream


def vb_decode(byte_stream):
    doc_ids = []
    num = 0
    for i in range(len(byte_stream)):
        if ord(byte_stream[i]) < 128:
            num = num * 128 + ord(byte_stream[i])
        else:
            num = num * 128 + (ord(byte_stream[i]) - 128)
            doc_ids.append(num)
            num = 0
    return doc_ids
```

可以通过命令 show_index word 来查看指定词的压缩编码：

```python
def print_vb_code(byte_stream):
    bytes_print = ''
    for i in range(len(byte_stream)):
        num = ord(byte_stream[i])
        one_bytestream = ''
        for j in range(8):
            one_bytestream += str(num % 2)
            num = int(num / 2)
        bytes_print += one_bytestream[::-1]
        bytes_print += ' '
    print(bytes_print)
```

3.4 构建轮排索引

为了进行通配符查询，有必要建立一个轮排索引表。通配符查询有以下一些种类：

1、尾通配符查询 mon*:找出所有包含以 mon 开头的词项的文档

2、首通配符查询*mon:找出所有包含以 mon 结尾的词项的文档

3、中间通配符查询 m*cine:找出所有包含以 m 开头并以 cine 结尾的词项的文档

```
191     # 创建轮排表
192     def create_Permuterm_index(self):
193         print('Begin creating Permuterm index.')
194         self.permuterm_index_table = sbst()
195         for item in self.table.keys():
196             word = item + '$'
197             for i in range(len(word)):
198                 self.permuterm_index_table.add([word[i:] + word[:i], item])
199         print('Finished creating Permuterm index. \n')
```

轮排索引表是倒排索引的一种特殊形式。在构建时，先在字符集中引入一个符号'$'以标识词项的结束。对于词项 hello， (hello$, ello$h, llohe,lohel,和 o$hell)是其轮排词汇集，每个扩展词都指向原始词项。

在该部分仅需构建轮排索引表即可，不涉及查询的步骤。构建轮排索引表时，为便于操作，采用 B-树（159 行）的数据结构，其部分内容如下图所示：

```
1   # -*- coding: utf-8 -*-
2   def _sbst_comparison(v1, v2):
3       """ Sorting function by default """
4       if v1[0] == v2[0]:
5           return 0
6       else:
7           return -1 if v1[0] < v2[0] else 1
8
9
10  def _sbst_simple_comparison(v1, v2):
11      """ Sorting function by default """
12      if v1 == v2:
13          return 0
14      else:
15          return -1 if v1 < v2 else 1
16
17
18  class _sbst_node:
19      """ Represents tree node """
20
21      def __init__(self, val, parent, direction=None):
22          self.level = self.len = 1
23          self.left = self.right = None
24          self.val = val
25          self.is_array = False
26          self.parent = parent
27          self.direction = direction
```

直接使用网络上现成的实现。至此，查询表已经建立完毕。

4、布尔查询

在倒排索引表建立完成后就可以编写布尔查询的相关代码：

```python
# 布尔查询，暂不支持显示文章摘要
def do_boolean_query(self, args):
    args = self.change_k(args)
    print('\nBoolean query.Does not support summary display temporarily.')
    print('\n')
    try:
        t1 = time.time()
        expression = args.replace('(', ' ( ').replace(')', ' ) ').split()
        doc_list = sorted(self.object.documents.keys())
        ret = self.object.indextable.boolean_query(expression, doc_list)
        t2 = time.time()
        if len(ret) == 0:
            print('Not found. (in {0:.5f} seconds)'.format(t2 - t1))
            if len(expression) == 1:
                self.object.indextable.correction(expression[0])
        else:
            if ret != 'Invalid boolean expression.':
                print('Total docs: {0} (in {1:.5f} seconds)'.format(len(ret), t2 - t1))
                print('Top-{0} rankings:\n'.format(min(self.k, len(ret))))
            cnt = 0
            for ID in ret:
                if cnt >= self.k:
                    break
                result = 'doc ID: {0} '.format(ID).ljust(12, ' ')
                result += 'doc name: {0}'.format(self.object.doc_lists[ID])
                print(result)
                cnt += 1
            print('\n')
    except Exception as e:
        print(e)
```

change_k 函数将检查用户是否更改目标输出数量，以及如果有更改的话用户想要将其更改为多少。

布尔查询主要包括 AND、NOT、OR 操作。其中 AND 相当于是求交集、OR 相当于求并集、NOT 相当于去重叠。

以 AND 操作为例，求交集时相当于对 AND 前后的两个词的倒排记录表做合并操作，其伪码如下所示：

```
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
```

两个倒排记录表的合并算法，摘自《信息检索导论》1.3 图 1-6

当然，实际上不必这么麻烦，python 可以很方便地对两个列表进行与或非的操作。

最差的布尔查询是依次对每个操作进行处理，然而实际上不同操作符之间通过一定的优先级定义就可以达到优化效率的效果。同时考虑到用户可能会任意组合三种操作，且可能用“(”和“)”来提高部分操作的优先级，因此采用栈结构来进行布尔查询的操作：

```
254     # 布尔查询 暂不支持中文
255     def boolean_query(self, words, doc_list):
256         priority = {'AND': 1, 'OR': 1, 'NOT': 2, '(': 0}
257         stack = []  # 只存单词不存操作
258         op = []  # 只存操作不存单词
259         ret = []
260         for i in range(0, len(words)):
261             if words[i] == 'AND' or words[i] == 'OR' or words[i] == 'NOT':
262                 while (len(op) > 0) and priority[op[len(op) - 1]] >= priority[words[i]]:
263                     stack = boolean_op(op.pop(), stack)
264                 op.append(words[i])
265             elif words[i] == '(':
266                 op.append('(')
267             elif words[i] == ')':
268                 while len(op) > 0 and op[len(op) - 1] != '(':
269                     stack = boolean_op(op.pop(), stack)
270                 op.pop()
271             else:
272                 # 列表的最后一项一定是空串'', 标识结束
273                 vec = vector_encode(self.table.get(words[i], [{}, 0])[0], doc_list)
274                 stack.append(vec)
275 
276         while len(op) > 0:
277             stack = boolean_op(op.pop(), stack)
278         if len(stack) > 1:
279             return 'Invalid boolean expression.'
280         res = stack[0]
281         ret = vector_decode(res, doc_list)
282         return ret
```

定义两个栈：stack 和 op，其中 stack 用于存取单词，op 用于存取操作。从左到右依次扫描，扫描到“(”时直接添加，扫描到“)”时则将 op 中的操作依次出栈对 stack 做操作，扫描到操作符时，如果当前操作符的优先级比栈顶操作符的优先级更高，则将 op 中的操作符出栈并对 stack 做操作。通过这样的规则来使得操作倒排表一点点减小，直到最后剩下一个最终结果。

对表的与或非操作的代码由 boolean_op 函数实现，其中定义了一系列匿名函数来实现与或非的操作：

```
14  def boolean_op(op, stack):
15      if op == 'AND':
16          vec1 = stack.pop()
17          vec2 = stack.pop()
18          res = list(map(lambda x, y: x and y, vec1, vec2))
19          stack.append(res)
20      elif op == 'OR':
21          vec1 = stack.pop()
22          vec2 = stack.pop()
23          res = list(map(lambda x, y: x or y, vec1, vec2))
24          stack.append(res)
25      elif op == 'NOT':
26          vec1 = stack.pop()
27          res = list(map(lambda x: not x, vec1))
28          stack.append(res)
29      return stack
30
```

在所有栈清空且检查后发现输入无误后即可进行解码返回结果操作（281 行）。由于之前得到的结果是若干个二级列表，通过解码将其返回为一个一级列表。

```python
def vector_encode(a, doc_list):
    return list(i in a for i in doc_list)


def vector_decode(v, doc_list):
    ret = []
    for i in range(len(v)):
        if v[i]:
            ret.append(doc_list[i])
    return ret
```

至此，布尔查询模块结束。

5、TF-IDF 值的计算

TF-IDF 是常用的加权技术，TF 是词频(Term Frequency)，表示词条（关键字）在文本中出现的频率；IDF 是逆文本频率指数(Inverse Document Frequency)，某一特定词语的 IDF 可以由总文件数目除以包含该词语的文件的数目（一般会将分母+1 以做平滑），再将得到的商取对数得到。

如果包含词条 t 的文档越少，IDF 越大，则说明词条具有很好的类别区分能力。其计算公式分别如下图所示：

$$\mathrm{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \qquad \text{词频(TF)} = \frac{\text{某个词在文章中的出现次数}}{\text{文章的总词数}}$$

$$\mathrm{idf}_i = \lg \frac{|D|}{|\{j : t_i \in d_j\}|} \qquad \text{逆文档频率(IDF)} = \log(\frac{\text{语料库的文档总数}}{\text{包含该词的文档数}+1})$$

TF-IDF 的值就是 TF 与 IDF 的乘积。

原理很简单，代码实现也很简单：

```python
    # 计算TF-IDF，暂不支持中文
    def compute_TFIDF(self, sentence_, language='en'):
        sentence = []
        if language == 'en':
            words = sentence_.split(' ')
            for item in words:
                if item:
                    sentence.append(item)
        elif language == 'zh':
            pass
        else:
            print("no support language")
            raise
        scores = {}
        sentence = Counter(sentence)
        for piece in sentence.items():
            doc_list = self.table[piece[0]]
            weight = (1 + math.log10(piece[1])) * math.log10(self.length / doc_list[1])
            for doc in doc_list[0].items():
                if scores.get(doc[0], 'null') != 'null':
                    scores[doc[0]] += (1 + math.log10(doc[1])) * math.log10(self.length / doc_list[1]) * weight
                else:
                    scores[doc[0]] = (1 + math.log10(doc[1])) * math.log10(self.length / doc_list[1]) * weight
        for i in scores.items():
            scores[i[0]] = scores[i[0]] / len(self.document_words[i[0]])
        scores = sorted(scores.items(), key=operator.itemgetter(1), reverse=True)
        return scores
```

6、通配符查询

前面提到，查询需要有一个轮排索引表，所以该模块会自动检查轮排索引表是否存在，若不存在则自动进行建立。建立的过程如前所述。

```
112     def do_wildcard_query(self, args):
113         args = self.change_k(args)
114         print('\nWildcard query.')
115         print('\n')
116         try:
117             t1 = time.time()
118             if not self.object.indextable.permuterm_index_table:
119                 self.object.indextable.create_Permuterm_index()
120             ret = self.object.indextable.find_regex_words(args)
121             words = ret
122             print('searched words: ', ret)
123             ret = self.object.indextable.compute_TFIDF(' '.join(ret))
124             t2 = time.time()
125             print('Total docs: {0} (in {1:.5f} seconds)'.format(len(ret), t2 - t1))
126             print('Top-{0} rankings:\n'.format(min(self.k, len(ret))))
```

在轮排索引表建立完成后就要查找正则词，前面提到了用$做词结束标识和用 B-树做轮排操作，基于此，在做通配符查询时，首先将每个通配查询词项进行旋转，使*出现在末尾，然后在轮排索引的 B-树中搜索，最后将搜索结果映射到常规词项。例如:

对于*X,查询 X$*；

对于*X*,查询 X*；

对于 X*Y,查询 Y$X*。

例如，对于 hel*o，查询 o$hel*；对于 fi*mo*er，先查 er$fi*，再除去不包含 mo 的词项。基于以上思想先做旋转操作，再将旋转后的结果与词典进行匹配，然后返回符合要求的候选词作为待查询结果。

```
201     # 查找正则词
202     def find_regex_words(self, _prefix):
203         prefix = _prefix + '$'
204         prefix = prefix[prefix.rindex('*') + 1:] + prefix[:prefix.index('*')]
205         candidates = []
206         for i in self.permuterm_index_table.forward_from(prefix):
207             if not i[0].startswith(prefix):
208                 break
209             candidates.append(i)
210         prefix = _prefix.split('*')
211         candidates_filterd = []
212         for _candidate in candidates:
213             seed = False
214             candidate = _candidate[1]
215             for pre in prefix:
216                 try:
217                     candidate = candidate[candidate.index(pre) + len(pre):]
218                 except:
219                     seed = True
220                     break
221             if not seed:
222                 candidates_filterd.append(_candidate[1])
223 
224         return candidates_filterd
```

之后就只需要根据候选测的倒排表查询对应文档及摘要信息即可。

7、短语查询

短语查询通过 phrase_query 部分实现，总体流程如下图所示：

```python
    # 短语查询
    def do_phrase_query(self, args):
        args = self.change_k(args)
        print('\nPhrase query.Does not support summary display temporarily.')
        print('\n')
        try:
            t1 = time.time()
            ret = self.object.indextable.phrase_query(args)
            scores = {}
            for i in ret:
                scores[i] = self.object.indextable.compute_TFIDF_with_docID(args, i)
            scores = sorted(scores.items(), key=operator.itemgetter(1), reverse=True)
            t2 = time.time()
            print('Total docs: {0} (in {1:.5f} seconds)'.format(len(scores), t2 - t1))
            print('Top-{0} rankings:\n'.format(min(self.k, len(scores))))
            for index, i in enumerate(scores):
                if index > self.k:
                    break
                hit_info = 'doc ID: {0} '.format(i[0]).ljust(12, ' ')
                hit_info += 'TF-IDF value: {0:.5f} '.format(i[1]).ljust(22, ' ')
                hit_info += 'doc name: {0}'.format(self.object.doc_lists[i[0]])
                print(hit_info)
                flag = show_summary(doc_list=self.object.doc_lists, index=i[0], word=args)
                if flag:
                    print('\n')
                # print(i)
        except Exception as e:
            print(e)
```

其中 phrase_query 部分的代码如下图所示：

```python
    def phrase_query(self, args, language='en'):
        sentence = []
        if language == 'en':
            words = args.split(' ')
            for item in words:
                if item:
                    sentence.append(item)
        elif language == 'zh':
            pass
        else:
            print("no support language")
            raise
        docs = []
        for i in range(len(sentence)-1):
            ret = self.table_2.get(sentence[i] + ' ' + sentence[i + 1], 'none')
            if ret == 'none':
                return []
            docs.append(set(ret[0].keys()))
        docs = set.intersection(*docs)
        docs_filterd = []
        for doc in docs:
            for i in range(len(self.document_words[doc])):
                seed = False
                if i == sentence[0]:
                    for index, token in enumerate(sentence):
                        if token != self.document_words[doc][i + index]:
                            seed = True
                            break
                if seed == False:
                    docs_filterd.append(doc)

        return docs_filterd
```

其实很简单，就是把短语做一下切词操作，然后对每个词的索引表做一次合并，如果存在未登录词则将其忽略掉，然后返回结果并对指定文档和指定词组中的每一个词求 TF-IDF 以获得相关性分数，并以此作为评级标准。

8、结果数目更改

用户通过—hits 参数可以更改结果的输出数目，通过自定义一个 change_k 函数来检测用户是否选择更改结果的输出数目（默认是 10）：

```python
    def change_k(self, args):
        k = self.k
        un_mattched = args.split(' ')
        hit_arg_rule = r'(?<=hits=)[\w]*'
        for item in un_mattched:
            res = re.search(hit_arg_rule, item)
            if res:
                un_mattched.remove(item)
                k_rule = r'(?<=hits=)[\d]*'
                k = re.search(k_rule, item).group()
                break
        args = ' '.join(un_mattched)
        try:
            tar = int(k)
            self.k = tar
            return args
        except Exception as e:
            print(e)
```

这部分代码实现很简单，通过一个正后发断言检测用户是否键入了—hits 参数，再通过另一个正后发断言来获取数字部分，最后将其转化为整数即可。

【实验结果分析】：

本查询系统目前仅支持对英文 txt 文本的信息检索工作。本部分将围绕以下 7 个模块依次予以解释和分析：用户交互、数据获取、查询表的建立、VB 编码查询、布尔查询、通配符查询、短语查询。

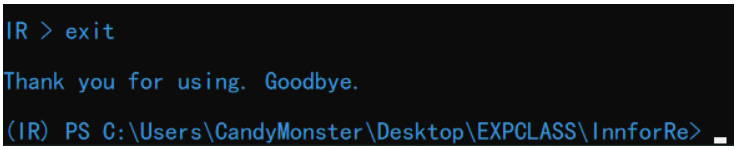
1、用户交互：

在命令行输入命令：python ./IRcmder.py，可以看到初始界面如下图所示：

```
(IR) PS C:\Users\CandyMonster\Desktop\EXPCLASS\InnforRe> python .\IRcmd.py


This is a simple Information Retrival System.
Copyright 2021 @CandyMonster37: https://github.com/CandyMonster37
A course final project for Information Retrival, and you can find the latest vertion of the codes here:
   https://github.com/CandyMonster37/InformationRetrival.git



                        Welcome to the Information Retrival System.


This is a simple Information Retrival System.
You can use some commands to do some work related to the information retrieval.
Only supports English.
Shell commands are defined internally.

Type 'help' or '?' to list all available commands.
Type 'help cmd' to see more details about the command 'cmd'.
Or type 'exit' to exit this system.
```

初始时会输出一些提示信息，包括代码所存储的 GitHub 仓库地址、关于本系统的一些简短的介绍，以及初始命令。输入 help 或者?可以查看命令列表：

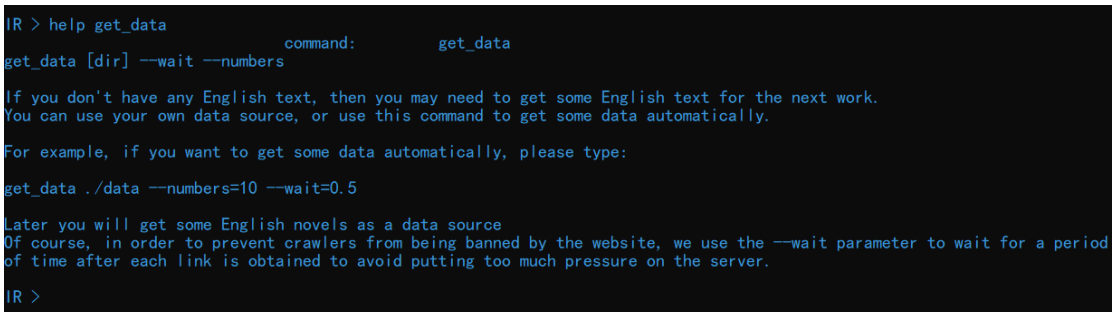
输入 exit 可以退出运行：



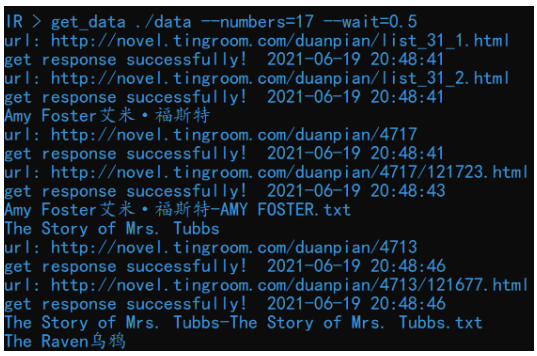
可以输入 help command 查看更详细的命令介介绍，这一点在之后的部分依次予以展示。

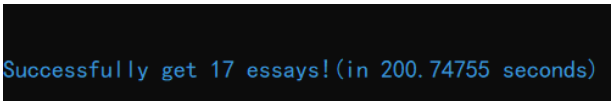
2、数据获取：

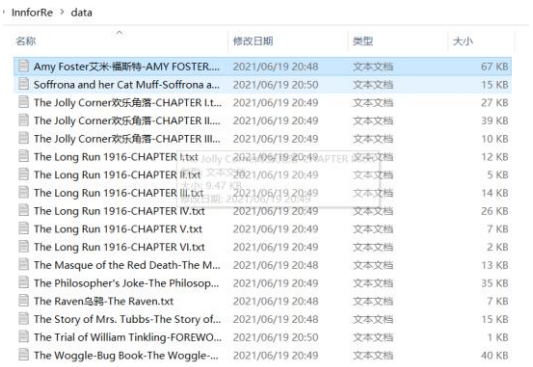
该部分功能通过 get_data 命令实现。输入 help get_data 查看关于该命令的详细介绍：



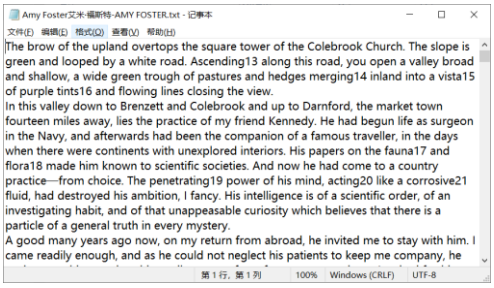
输入命令 get_data ./data –numbers=17 –wait=0.5 可以从内置的某英文小说网站获取 17 篇短篇小说并保存到./data 目录下。每获取一个链接，爬虫都会等待 0.5s，以防止被服务器封停。爬取结果如下图所示：







部分内容如下图所示：

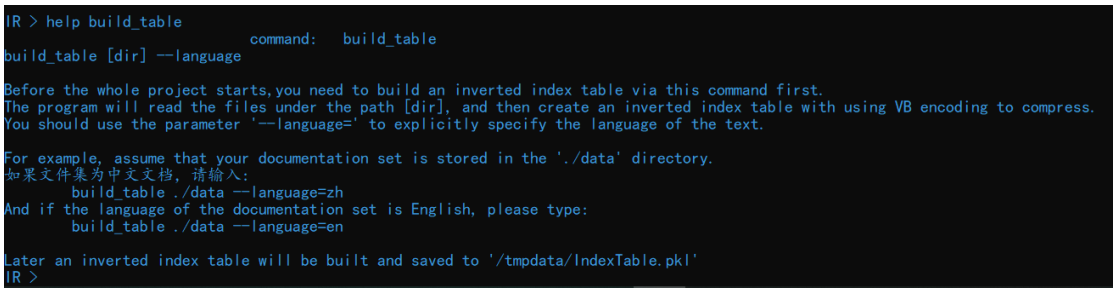


可以看到爬虫的运行花费了较长的时间，这是因为爬虫实际访问的链接数大于保存的文本数。在爬取过程中，可能由于网络问题也可能由于服务器的问题等，可能会出现资源访问失败的情况（如下图所示）：

因此耽误了一些时间。

3、查询表的建立

查询表部分的功能通过 build_table 命令实现。输入 help build_table 查看关于该命令的详细介绍：



输入命令 build_table ./data 可以从./data 目录下读取文件并建立查询表。

```
IR > build_table ./data
Begin loading and build index.
Finished loading and build index.

Begin creating Permuterm index.
Finished creating Permuterm index.

IR > _
```

整个过程并不会花费太多时间。

4、查看指定词的 VB 编码

可以使用命令 show_index 查看指定词的 VB 压缩码。输入 help show_index 查看关于该命令的详细介绍：

```
IR > help show_index
                        command:  show_index
show_index [word]

When builting the index table, I use VB code to compress.
After building the index table, you can view the VB compression code of the word you want via this command.

For example, if you want to see the VB compression code of the word 'we', please type:

show_index we

Later the screen will show the VB compression code of 'we'

IR > _
```

输入命令 show_index we 可以查看单词 we 的 VB 压缩码：

```
IR > show_index we
[b'\x80', b'\x81', b'\x85', b'\x89', b'\x8b', b'\x8d', b'\x8f', b'\x91', b'\x93', b'\x95', b'\x97', b'\x99', b'\x9b', b'\x9e']
10000000 10000001 10000101 10001001 10001011 10001101 10001111 10010001 10010011 10010101 10010111 10011001 10011011 10011110


IR > show_index he
[b'\x80', b'\x82', b'\x85', b'\x87', b'\x89', b'\x8b', b'\x8d', b'\x8f', b'\x92', b'\x95', b'\x97', b'\x99', b'\x9b', b'\x9e']
10000000 10000010 10000101 10000111 10001001 10001011 10001101 10001111 10010010 10010101 10010111 10011001 10011011 10011110


IR > show_index what
[b'\x80', b'\x81', b'\x83', b'\x85', b'\x87', b'\x89', b'\x8b', b'\x8d', b'\x8f', b'\x91', b'\x93', b'\x95', b'\x97', b'\x99', b'\x9b', b'\x9e']
10000000 10000001 10000011 10000101 10000111 10001001 10001011 10001101 10001111 10010001 10010011 10010101 10010111 10011001 10011011 10011110
```

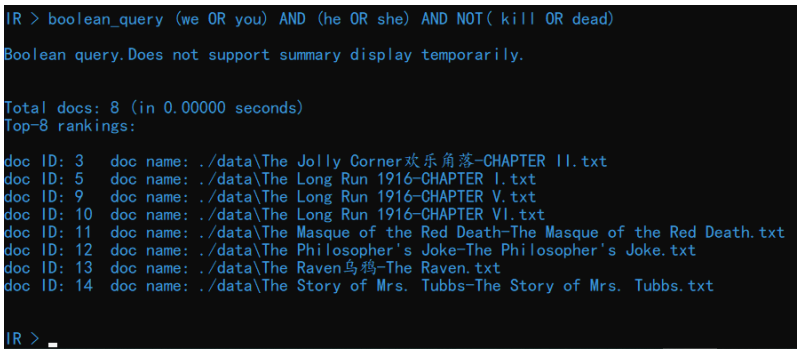
这里分别测试了单词 we、he、what。

5、布尔查询

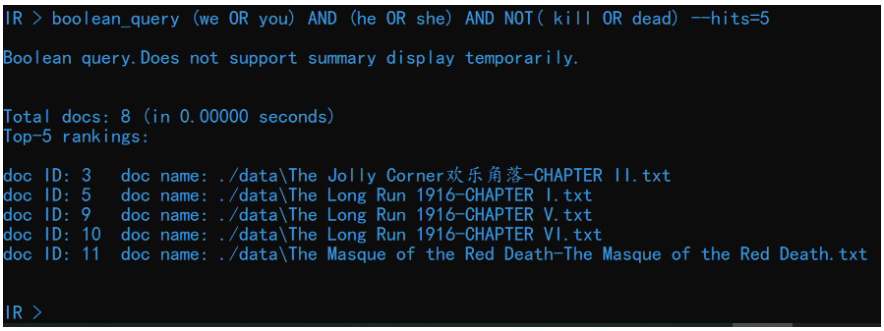
可以使用命令 boolean_query 对指定操作进行布尔查询。输入 help boolean_query 查看关于该命令的详细介绍：

```
IR > help boolean_query
                              command:    boolean_query
boolean_query [options] --hit

After creating the index table, you can use this command for boolean query.
Available operations are AND, OR and NOT, and you can use () to combine them arbitrarily.
For example, if you want to find articles that contain 'we' and 'are' but not 'you', please type:

boolean_query we AND are NOT you --hits=7

Later the screen will show some articles which are found.
Only supports English.

IR > _
```

输入命令 boolean_query (we OR you) AND (he OR she) AND NOT(kill OR dead)可以查看包含 we 或者 you、且包含 he 或者 she、且不包含 kill 或者 dead 的文档：



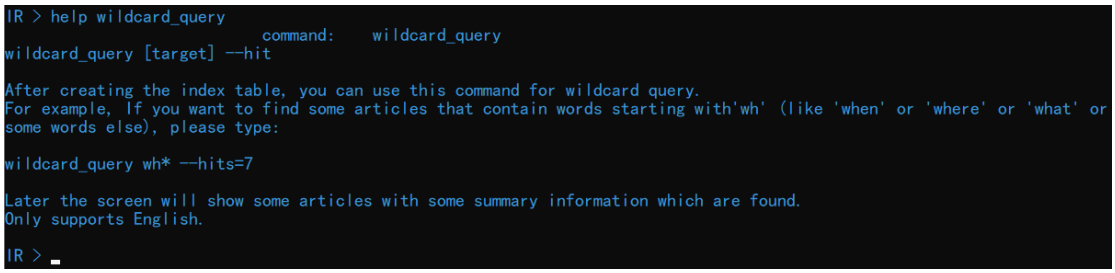
速度还是比较快的，小数点后 5 位都捕捉不到时间的变化。加入—hits 参数可以限制结果的展示数量，如输入命令 boolean_query (we OR you) AND (he OR she) AND NOT(kill OR dead) –hits=5：



可以看到共有 8 篇文档满足要求，只展示了前 5 篇文档。

6、通配符查询

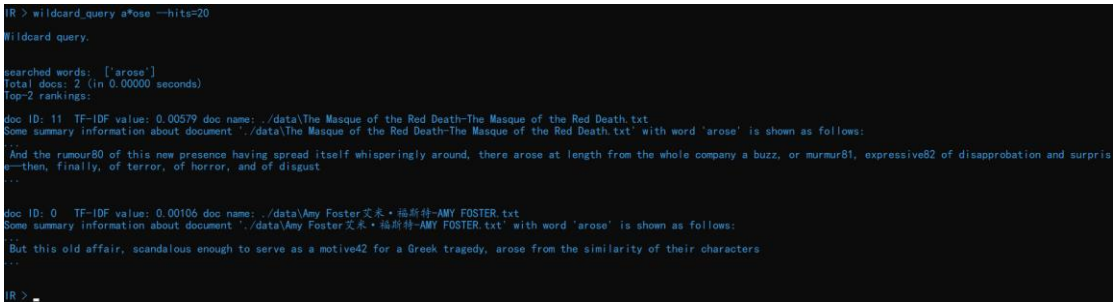
使用命令 wildcard_query 对指定缺省词进行通配符查询，输入 help wildcard_query 查看关于该命令的详细介绍：



输入命令 wildcard_query a*e 可以查看包含以 a 开头以 e 结尾的单词的文档：

可以看到其候选词有很多，如'able', 'above', 'absence', 'absolute'等。总共找到了 16 篇符合要求的文档，这里输出了前 5 篇（这里是以输出的文档个数为标准计数的，同一篇文档、不同词汇也会累加）。

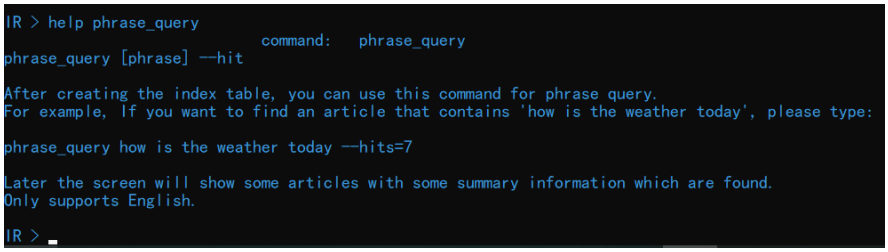
加入—hits 参数可以改变结果的显示数量，如 wildcard_query a*ose --hits=20 就可以显示包含 a*ose 单词的前 20 份结果：



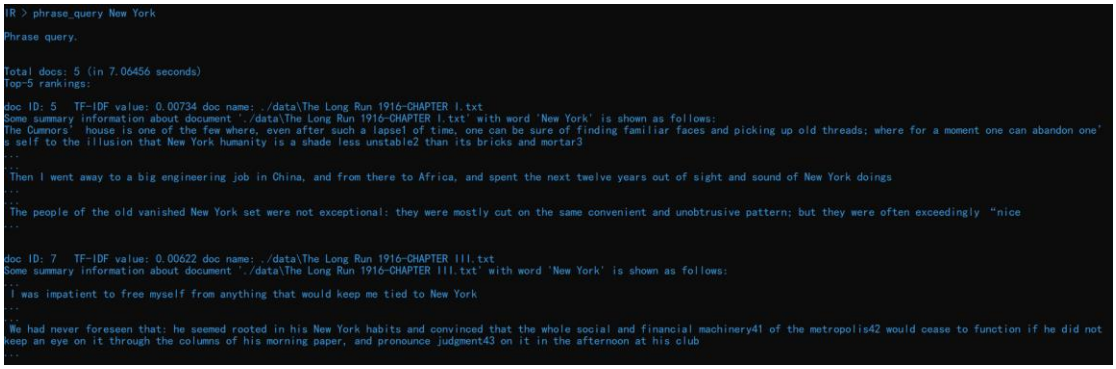
当然，如果文档没有那么多的话只会显示找到的文档。

7、短语查询

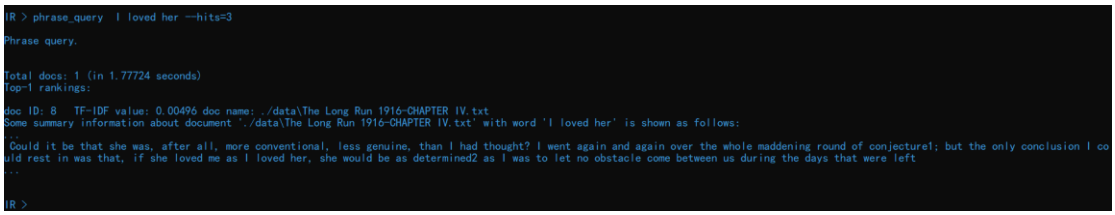
短语查询需要使用命令 phrase_query，从而对指定短语进行查询，输入 help phrase_query 查看关于该命令的详细介绍：



输入 phrase_query New York 可以查看包含 New York 的文档及摘要信息：



同样地可以通过—hits 参数限制输出的结果数目，如输入命令 phrase_query　I loved her --hits=3 可以显示最多 3 篇包含短语 I loved her 的文档：



当然，同样会受到实际文档数目的限制。

【实验总结】：

总体来讲，本项目已经基本实现了针对文档的布尔查询、通配符查询、词组查询等功能，然而还是略有改进之处：

1、不支持单个词查找。这部分是我的疏忽，因为一直觉得是一个很简单的功能所以一直就没有做，直到写实验报告的时候才发现忘了实现这个功能，然而时间上已经来不及了。其实实现很简单，只需要直接查倒排表中的指定词项即可获取对应的文档 ID，然后依次输出即可。

2、文档摘要输出不够美观。这部分我一直在调试，尝试更换不同的输出格式，仍然没办法做到整齐、简约，这是我自己审美有一点点问题。

3、目前仅支持对英文文档的操作。其实我在做的时候有考虑对中文文档做检索，想到 jieba 可以很容易地分词，进而也应该可以做信息检索工作。然而实际在实现的时候仍然有些理不清楚，中文的词与词之间、词与字之间的界限并不像英文那样明晰，做到什么样的截取能卡到一个刚好的边界，我实在是拿捏不准，因此暂时将其放弃，也希望能得到更好的指导。

4、未设计评价算法。一般而言，对于一个信息检索系统，需要有一个评价算法来评价

其好坏，也需要一个标准测试集去验证它的好坏，这部分测试我迫于时间原因还未来得及做。 除去以上需要改进的地方，本项目也有一定的可圈可点之处，如：在英文文本上做检索的准确率较高、处理性能较好、能自动批量化获取数据、交互性强等。 此外，通过本次项目，我也明白了信息检索工作中的一些基本方法和基本原理，也提高了自己的代码能力和资料查找能力。信息检索是一门涉及面很广的学科，我目前所掌握的仅如海滩上的一湾小水坑，我仍有很多需要学习的地方。
评语及评分（指导教师）
【评语】： **评分：** **日期：**

附件：

实验报告说明

1．实验名称：要用最简练的语言反映实验的内容。

2．实验目的：目的要明确，要抓住重点。

3．实验环境：实验用的软硬件环境（配置）。

4．实验方案设计（思路、步骤和方法等）：这是实验报告极其重要的内容。包括概要设计、详细设计和核心算法说明及分析，系统开发工具等。应同时提交程序或设计电子版。

对于**设计型和综合型实验**，在上述内容基础上还应该画出流程图、设计思路和设计方法，再配以相应的文字说明。

对于**创新型实验**，还应注明其创新点、特色。

5．实验结果分析：即根据实验过程中所见到的现象和测得的数据，进行对比分析并做出结论（可以将部分测试结果进行截屏）。

6．实验总结：对本次实验的心得体会，所遇到的问题及解决方法，其他思考和建议。

7．评语及评分：指导教师依据学生的实际报告内容，用简练语言给出本次实验报告的评价和价值。