山东大学<u>　　计算机科学与技术　　　</u>学院

<u>信息检索与数据挖掘</u> 课程实验报告

学号： 201600130053	**姓名：**　　王斌	**班级：**　16 人工智能

实验题目： 对推特文本数据建立 inverted index 的 postings 并在此基础上完成 Boolean 查询等功能（功能可自选拓展）

实验内容：

#Homework3: Inverted index and Boolean Retrieval Model

- **任务：**
  使用我们介绍的方法，在 tweets 数据集上构建 inverted index;
  实现 Boolean Retrieval Model，使用 TREC 2014 test topics 进行测试；
- Boolean Retrieval Model：
  Input：a query (like Ron Weasley birthday)
  Output: print a list of top k relevant tweets.
  支持 and, or ,not；查询优化可以选做；
- **注意：**
  对于 tweets 与 queries 使用相同的预处理；

实验环境：

Spyder+python3.6
Win10

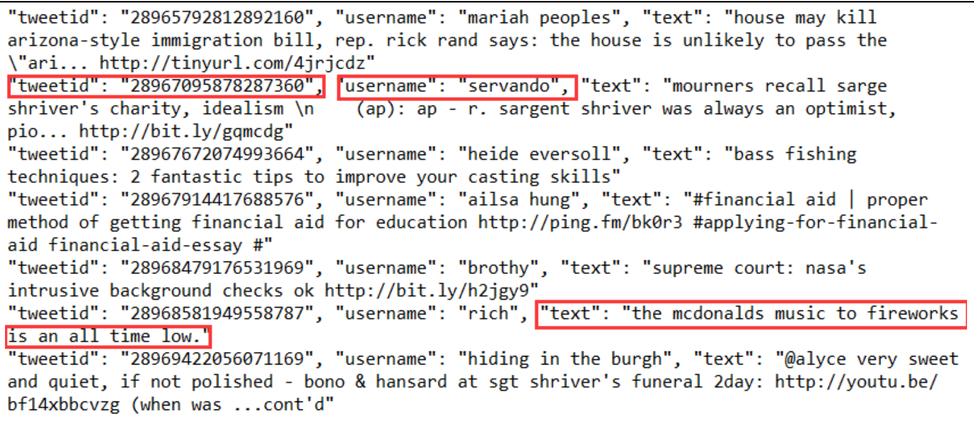
实验过程中遇到和解决的问题：

（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

一、　对推特数据的处理

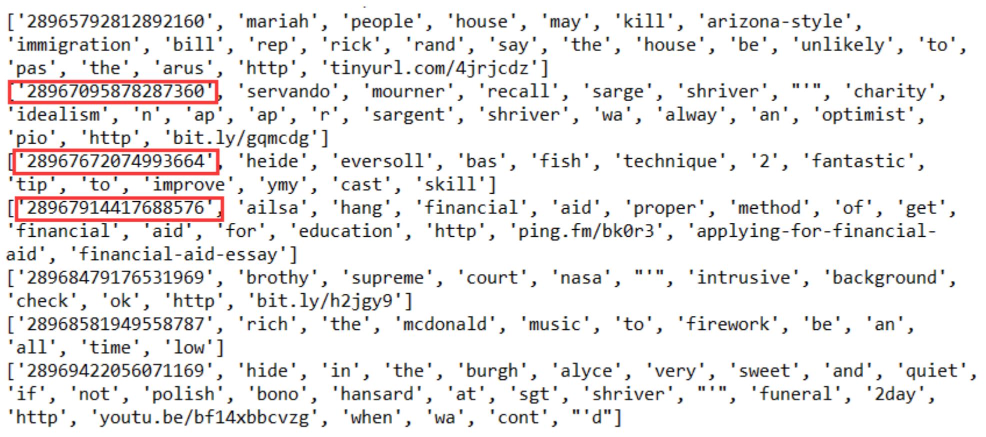
打开推特的文本数据发现数据具有较好的结构性，信息主要有 userName、clusterNo、text、timeStr、tweetId、errorCode、textCleaned、relevance 这些部分的信息，但是除了红色标注的，对于我们的检索任务而言，其它的都是冗余的，我们首先需要集中提取出红色的三部分信息来建立 inverted index 的 postings。

按行读取每条 tweet 后调用 tokenize_tweet 方法对其进行处理，并进行分词后对单词的统一变小写、单复数和动词形式统一等处理，使用 TextBlob 工具包，处理后的推特如下所示：



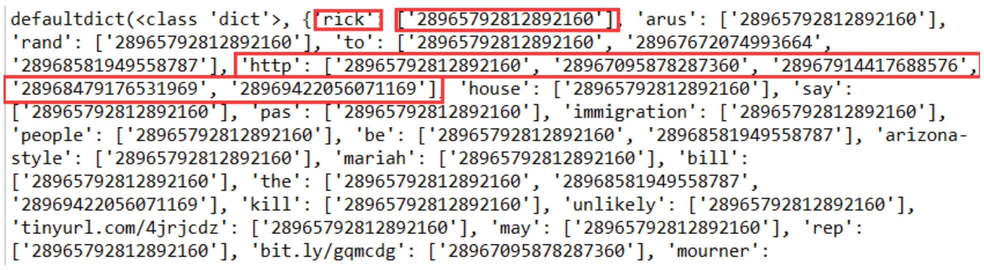
```
"tweetid": "28965792812892160", "username": "mariah peoples", "text": "house may kill
arizona-style immigration bill, rep. rick rand says: the house is unlikely to pass the
\"ari... http://tinyurl.com/4jrjcdz"
"tweetid": "28967095878287360", "username": "servando", "text": "mourners recall sarge
shriver's charity, idealism \n    (ap): ap - r. sargent shriver was always an optimist,
pio... http://bit.ly/gqmcdg"
"tweetid": "28967672074993664", "username": "heide eversoll", "text": "bass fishing
techniques: 2 fantastic tips to improve your casting skills"
"tweetid": "28967914417688576", "username": "ailsa hung", "text": "#financial aid | proper
method of getting financial aid for education http://ping.fm/bk0r3 #applying-for-financial-
aid financial-aid-essay #"
"tweetid": "28968479176531969", "username": "brothy", "text": "supreme court: nasa's
intrusive background checks ok http://bit.ly/h2jgy9"
"tweetid": "28968581949558787", "username": "rich", "text": "the mcdonalds music to fireworks
is an all time low."
"tweetid": "28969422056071169", "username": "hiding in the burgh", "text": "@alyce very sweet
and quiet, if not polished - bono & hansard at sgt shriver's funeral 2day: http://youtu.be/
bf14xbbcvzg (when was ...cont'd"
```

然后进行分词等处理后的推特如下：



```
['28965792812892160', 'mariah', 'people', 'house', 'may', 'kill', 'arizona-style',
'immigration', 'bill', 'rep', 'rick', 'rand', 'say', 'the', 'house', 'be', 'unlikely', 'to',
'pas', 'the', 'arus', 'http', 'tinyurl.com/4jrjcdz']
['28967095878287360', 'servando', 'mourner', 'recall', 'sarge', 'shriver', "'", 'charity',
'idealism', 'n', 'ap', 'ap', 'r', 'sargent', 'shriver', 'wa', 'alway', 'an', 'optimist',
'pio', 'http', 'bit.ly/gqmcdg']
['28967672074993664', 'heide', 'eversoll', 'bas', 'fish', 'technique', '2', 'fantastic',
'tip', 'to', 'improve', 'ymy', 'cast', 'skill']
['28967914417688576', 'ailsa', 'hang', 'financial', 'aid', 'proper', 'method', 'of', 'get',
'financial', 'aid', 'for', 'education', 'http', 'ping.fm/bk0r3', 'applying-for-financial-
aid', 'financial-aid-essay']
['28968479176531969', 'brothy', 'supreme', 'court', 'nasa', "'", 'intrusive', 'background',
'check', 'ok', 'http', 'bit.ly/h2jgy9']
['28968581949558787', 'rich', 'the', 'mcdonald', 'music', 'to', 'firework', 'be', 'an',
'all', 'time', 'low']
['28969422056071169', 'hide', 'in', 'the', 'burgh', 'alyce', 'very', 'sweet', 'and', 'quiet',
'if', 'not', 'polish', 'bono', 'hansard', 'at', 'sgt', 'shriver', "'", 'funeral', '2day',
'http', 'youtu.be/bf14xbbcvzg', 'when', 'wa', 'cont', "'d"]
```

最后再构建 postings，采用字典结构，其中将每个单词作为键值，后面跟着包含该单词的 tweet 的 tweetid 列表。



```
defaultdict(<class 'dict'>, {'rick': ['28965792812892160'], 'arus': ['28965792812892160'],
'rand': ['28965792812892160'], 'to': ['28965792812892160', '28967672074993664',
'28968581949558787'], 'http': ['28965792812892160', '28967095878287360', '28967914417688576',
'28968479176531969', '28969422056071169'], 'house': ['28965792812892160'], 'say':
['28965792812892160'], 'pas': ['28965792812892160'], 'immigration': ['28965792812892160'],
'people': ['28965792812892160'], 'be': ['28965792812892160', '28968581949558787'], 'arizona-
style': ['28965792812892160'], 'mariah': ['28965792812892160'], 'bill':
['28965792812892160'], 'the': ['28965792812892160', '28968581949558787',
'28969422056071169'], 'kill': ['28965792812892160'], 'unlikely': ['28965792812892160'],
'tinyurl.com/4jrjcdz': ['28965792812892160'], 'may': ['28965792812892160'], 'rep':
['28965792812892160'], 'bit.ly/gqmcdg': ['28967095878287360'], 'mourner':
```

二、 对查询的输入进行处理

注意需要对查询进行和 tweet 同样的分词等处理，保持一致性，主要代码如下所示：

```
 terms=TextBlob(document).words.singularize()
     result=[]
     for word in terms:
         expected_str = Word(word)
         expected_str = expected_str.lemmatize("v")
         if expected_str not in uselessTerm:
                result.append(expected_str)
        return result
```

主要是对输入的查询进行语义逻辑的识别，判断是什么样的布尔查询，在本次实验中，针对单个 and、or、not（A and B、A or B、A not B）三种布尔查询进行了实现，并在此基础上对双层逻辑的如 A and B and C、A or B or C、(A and B) or C、(A or B) and C 的实现，并作为功能拓展实现了对一般输入语句进行的排序查询，可以返回排序最靠前的若干个结果。如下所示：(用查询的单词在该文档中出现的个数/总数作为简单的排序分数)

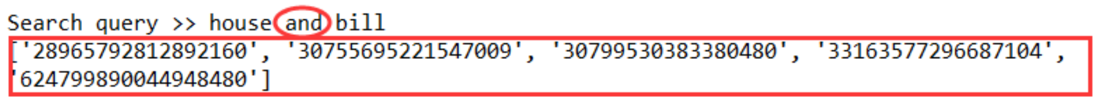
```
def do_search():
    terms = token(input("Search query >> "))
    if terms == []:
        sys.exit()
    #搜索的结果答案

    if len(terms)==3:
        #A and B
        if terms[1]=="and":
            answer = merge2_and(terms[0],terms[2])
        #A or B
        elif terms[1]=="or":
            answer = merge2_or(terms[0],terms[2])
        #A not B
        elif terms[1]=="not":
            answer = merge2_not(terms[0],terms[2])
        #输入的三个词格式不对
        else:
            print("input wrong!")

    elif len(terms)==5:
        #A and B and C
        if (terms[1]=="and") and (terms[3]=="and"):
            answer = merge3_and(terms[0],terms[2],terms[4])
            print(answer)
        #A or B or C
        elif (terms[1]=="or") and (terms[3]=="or"):
            answer = merge3_or(terms[0],terms[2],terms[4])
            print(answer)
        #(A and B) or C
        elif (terms[1]=="and") and (terms[3]=="or"):
            answer = merge3_and_or(terms[0],terms[2],terms[4])
            print(answer)
        #(A or B) and C
        elif (terms[1]=="or") and (terms[3]=="and"):
            answer = merge3_or_and(terms[0],terms[2],terms[4])
            print(answer)
        else:
            print("More format is not supported now!")
    #进行自然语言的排序查询，返回按相似度排序的最靠前的若干个结果
    else:
        leng = len(terms)
        answer = do_rankSearch(terms)
        print ("[Rank_Score: Tweetid]")
        for (tweetid,score) in answer:
            print (str(score/leng)+": "+tweetid)
```

其中 merge 合并列表时采用同时遍历的方法，降低复杂度为 O(x+y)，如下 merge2_and 所示：

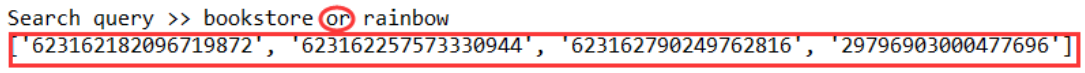
```
def merge2_and(term1,term2):
    global postings
    answer = []
    if (term1 not in postings) or (term2 not in postings):
        return answer
    else:
        i = len(postings[term1])
        j = len(postings[term2])
        x=0
        y=0
        while x<i and y<j:
            if postings[term1][x]==postings[term2][y]:
                answer.append(postings[term1][x])
                x+=1
                y+=1
            elif postings[term1][x] < postings[term2][y]:
                x+=1
            else:
                y+=1
        return answer
```

三、　查询测试展示

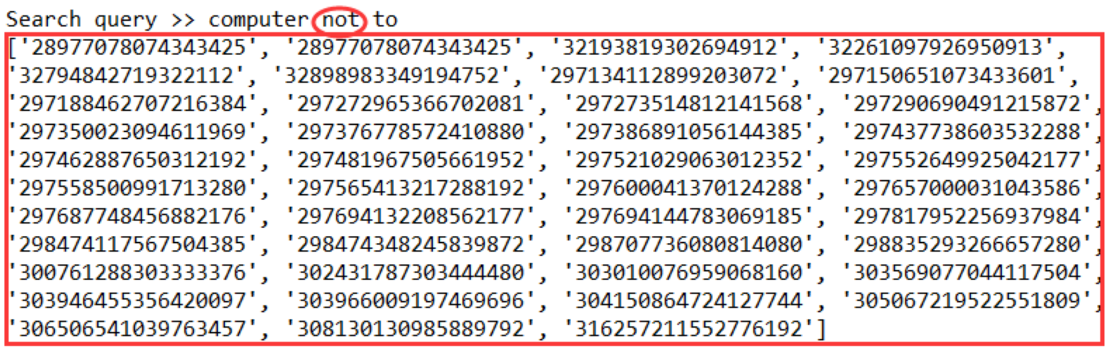
1、A and B、A or B、A not B:



```
Search query >> house and bill
['28965792812892160', '30755695221547009', '30799530383380480', '33163577296687104',
'624799890044948480']
```

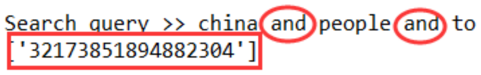


```
Search query >> bookstore or rainbow
['623162182096719872', '623162257573330944', '623162790249762816', '29796903000477696']
```

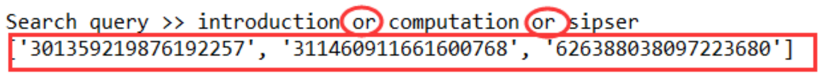


```
Search query >> computer not to
['28977078074343425', '28977078074343425', '32193819302694912', '32261097926950913',
'32794842719322112', '32898983349194752', '297134112899203072', '297150651073433601',
'297188462707216384', '297272965366702081', '297273514812141568', '297290690491215872',
'297350023094611969', '297376778572410880', '297386891056144385', '297437738603532288',
'297462887650312192', '297481967505661952', '297521029063012352', '297552649925042177',
'297558500991713280', '297565413217288192', '297600041370124288', '297657000031043586',
'297687748456882176', '297694132208562177', '297694144783069185', '297817952256937984',
'298474117567504385', '298474348245839872', '298707736080814080', '298835293266657280',
'300761288303333376', '302431787303444480', '303010076959068160', '303569077044117504',
'303946455356420097', '303966009197469696', '304150864724127744', '305067219522551809',
'306506541039763457', '308130130985889792', '316257211552776192']
```

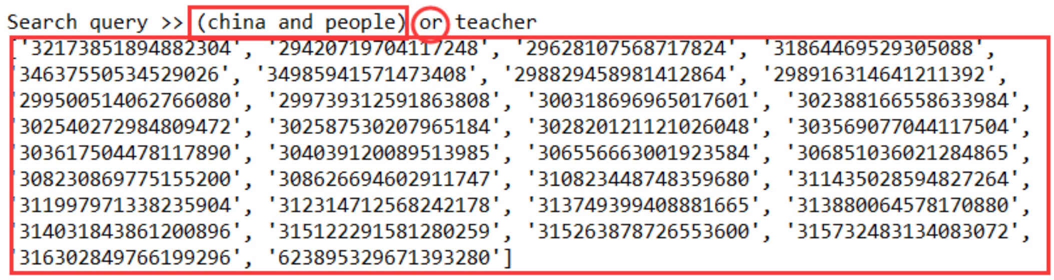
2、A and B and C、A or B or C、(A and B) or C、(A or B) and C



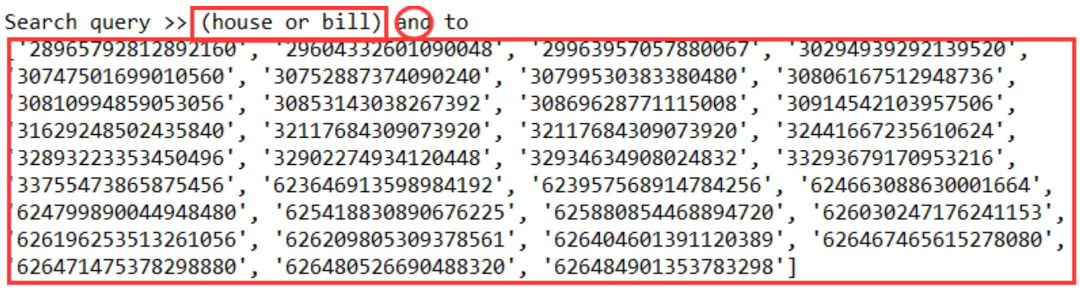
```
Search query >> china and people and to
['32173851894882304']
```



```
Search query >> introduction or computation or sipser
['301359219876192257', '311460911661600768', '626388038097223680']
```

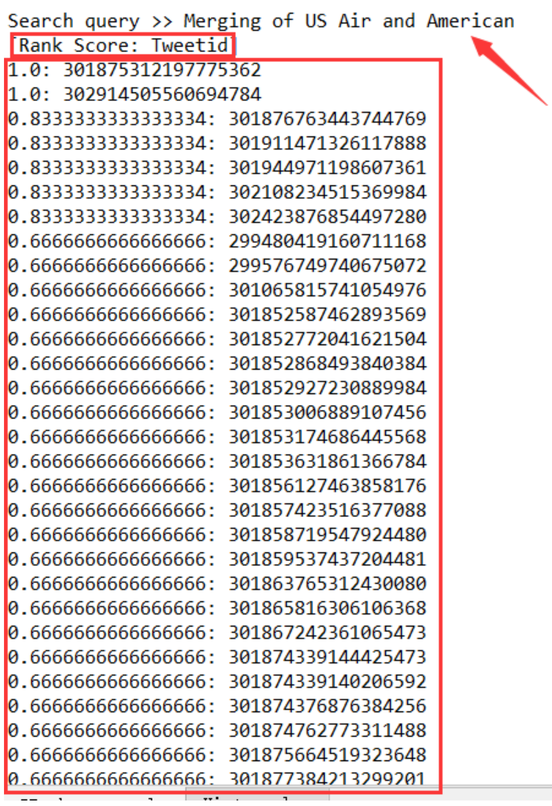


```
Search query >> (china and people) or teacher
['32173851894882304', '29420719704117248', '29628107568717824', '31864469529305088',
'34637550534529026', '34985941571473408', '298829458981412864', '298916314641211392',
'299500514062766080', '299739312591863808', '300318696965017601', '302388166558633984',
'302540272984809472', '302587530207965184', '302820121121026048', '303569077044117504',
'303617504478117890', '304039120089513985', '306556663001923584', '306851036021284865',
'308230869775155200', '308626694602911747', '310823448748359680', '311435028594827264',
'311997971338235904', '312314712568242178', '313749399408881665', '313880064578170880',
'314031843861200896', '315122291581280259', '315263878726553600', '315732483134083072',
'316302849766199296', '623895329671393280']
```

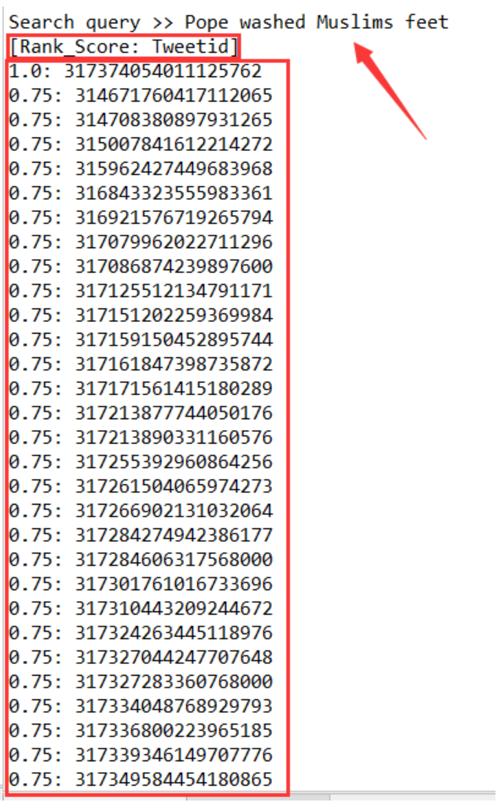


```
Search query >> (house or bill) and to
['28965792812892160', '29604332601090048', '29963957057880067', '30294939292139520',
'30747501699010560', '30752887374090240', '30799530383380480', '30806167512948736',
'30810994859053056', '30853143038267392', '30869628771115008', '30914542103957506',
'31629248502435840', '32117684309073920', '32117684309073920', '32441667235610624',
'32893223353450496', '32902274934120448', '32934634908024832', '33293679170953216',
'33755473865875456', '623646913598984192', '623957568914784256', '624663088630001664',
'624799890044948480', '625418830890676225', '625880854468894720', '626030247176241153',
'626196253513261056', '626209805309378561', '626404601391120389', '626467465615278080',
'626471475378298880', '626480526690488320', '626484901353783298']
```

3、一般的语句查询：



```
Search query >> Merging of US Air and American
[Rank Score: Tweetid]
1.0: 301875312197775362
1.0: 302914505560694784
0.8333333333333334: 301876763443744769
0.8333333333333334: 301911471326117888
0.8333333333333334: 301944971198607361
0.8333333333333334: 302108234515369984
0.8333333333333334: 302423876854497280
0.6666666666666666: 299480419160711168
0.6666666666666666: 299576749740675072
0.6666666666666666: 301065815741054976
0.6666666666666666: 301852587462893569
0.6666666666666666: 301852772041621504
0.6666666666666666: 301852868493840384
0.6666666666666666: 301852927230889984
0.6666666666666666: 301853006889107456
0.6666666666666666: 301853174686445568
0.6666666666666666: 301853631861366784
0.6666666666666666: 301856127463858176
0.6666666666666666: 301857423516377088
0.6666666666666666: 301858719547924480
0.6666666666666666: 301859537437204481
0.6666666666666666: 301863765312430080
0.6666666666666666: 301865816306106368
0.6666666666666666: 301867242361065473
0.6666666666666666: 301874339144425473
0.6666666666666666: 301874339140206592
0.6666666666666666: 301874376876384256
0.6666666666666666: 301874762773311488
0.6666666666666666: 301875664519323648
0.6666666666666666: 301877384213299201
```



```
Search query >> Pope washed Muslims feet
[Rank_Score: Tweetid]
1.0: 317374054011125762
0.75: 314671760417112065
0.75: 314708380897931265
0.75: 315007841612214272
0.75: 315962427449683968
0.75: 316843323555983361
0.75: 316921576719265794
0.75: 317079962022711296
0.75: 317086874239897600
0.75: 317125512134791171
0.75: 317151202259369984
0.75: 317159150452895744
0.75: 317161847398735872
0.75: 317171561415180289
0.75: 317213877744050176
0.75: 317213890331160576
0.75: 317255392960864256
0.75: 317261504065974273
0.75: 317266902131032064
0.75: 317284274942386177
0.75: 317284606317568000
0.75: 317301761016733696
0.75: 317310443209244672
0.75: 317324263445118976
0.75: 317327044247707648
0.75: 317327283360768000
0.75: 317334048768929793
0.75: 317336800223965185
0.75: 317339346149707776
0.75: 317349584454180865
```

结论分析与体会：

根据 inverted index 的模型完成了布尔的查询的要求，复杂的布尔查询也可以在基本的 and、or、not 逻辑基础实现上嵌套实现，最后通过用查询的单词在该文档中出现的个数/总数作为简单的排序检索比较简陋，结果需要进一步评估，在本次 inverted index 模型中没有考虑 TF、IDF 和文档长度等信息，还需要进一步完善来满足更高级的应用需求。