山东师范大学

《信息检索技术》

课程设计报告

题　　目：基于倒排索引和布尔检索的搜索引擎设计

学生姓名：隋　远

学　　号：201911010105

专　　业：计算机科学与技术(非师范)

指导教师：嵇　存

学　　院：信息科学与工程学院

2021 年 11 月 5 日

（一） 项目简介

本项目基于给定的数据集，实现一个检索系统。

（二） 需求分析

实验要求如下：

1. 基于给定数据集建立倒排索引

2. 进行布尔检索

3. 实现检索结果排序

4. 添加拼写校对

5. 进行索引压缩

6. 实现快速检索

7. 索引更新策略

信息检索系统除完成基本检索功能外，还应具有一定的鲁棒性，即针对用户的错误输入或信息不充分等特殊情况具有一定的解决方案。另一方面，为了尽可能地提高信息检索系统的效率，应当充分考虑索引的存储与更新问题。

（三） 概要设计

本系统设计遵守一般信息检索系统构建方法，主要包括语言分析器、索引器、文本查询分析器、拼写校对、索引压缩器、评分排序等模块。以下为各个模块的简要概述：

1. 语言分析器（preProcess.py）：对原始文档信息进行处理，实现去除数字、标点符号、停用词，提取题干等操作，并返回文档经过预处理后的分词列表。

2. 索引器（establishIndex.py）：使用层次型位置倒排索引方法生成索引表，

获得倒排索引表以及词项列表，并将结果储存到./cache 文件下，方便后续查询中直接从磁盘调用到内存上，用空间换取时间效率。

3. 文本查询分析器（boolRetrieval.py）：使用布尔检索在索引表中查找与用户查询最接近的结果，并调用（operateDocList.py）完成索引的交、并、差等操作。

4. 拼写校对（spellingCheck.py）：对文本查询分析器的结果做进一步的拼写校对后再在索引表中进行检索并返回与用户查询最接近的结果，使用基于字符串的编辑距离进行拼写校正（使用 Big.txt 作为单词列表）。

5. 索引压缩器（compressIndex.py）：使用可变长字节编码和 gap-encoding 的方法对索引列表进行 encode 处理，在结果输出时再通过 decode 模块还原到真实结果，实现空间压缩。

6. 评分排序：用于实现检索结果的排序，已知的评分函数有余弦相似度、静态得分、近邻性等，使用评分组合进行检索排序。

（四） 详细设计

4.1 语言分析器

语言分析器需要对原始文档信息进行处理得到提纯后的分词列表，进行的操作主要包括去除数字（因原始文档中包含大量邮件信息，为保证检索效果，去除这部分数字；如果想对信息进行高级检索，如检索邮件时间周期等，可对这一部分数据做进一步处理）、标点符号、停用词、提取题干等操作。

```
class PreprocessFile:
    def __init__(self):
        pass

    def preProcess(self, filePath):
        file = open(filePath, 'r')
        doc = file.read()
        doc_ = self.stemming(doc)  # doc_返回文档经预处理后的分词列表
        return doc_
```

Figure 1 PreprocessFile 类

将语言分析器封装在 PreprocessFile 类中，调用 preProcess 返回文档经过处理后的分词列表；stemming 方法中封装文档预处理方法，完成去除数字、标点符号、停用词（下载 nltk.stopwords）、分词以及提取词干操作。

```
def stemming(self, doc):
    """
    :param doc: 读取得到的文档内容
    :return: doc_ 返回文档经过文本处理后的分词列表
    """
    lower = doc.lower()
    deleteSignal = [',', '.', ';', '-', '&', ':', '<', '@', '>', "'", '--', '$', '`', '(', ')', '+', '!', '*', '"',
                    '?', '#', '*', '%', '~']
    # 去除数字
    lower = re.sub(r'[0-9]+', '', lower)
    # 去除标点符号
    for signal in deleteSignal:
        lower = lower.replace(signal, '')
    without_punctuation = lower
    # 分词
    tokens = nltk.word_tokenize(without_punctuation)
    # 去除停用词
    without_stopwords = [w for w in tokens if not w in stopwords.words('english')]
    # 提取题干,e.g., cleaning/cleans/cleaned等==clean
    cleaner = nltk.stem.SnowballStemmer('english')
    cleaned_text = [cleaner.stem(ws) for ws in without_stopwords]
    return cleaned_text
```

Figure 2 Stemming 方法封装文档预处理方法

4.2 索引器

索引器用于构建文档的倒排索引表，为提高效率，将构建得到的索引表存储到磁盘上，检索时从磁盘上调用到内存中。

```
class EstablishIndex:
    """
    filePath = "./hyatt-k"
    load_data: 获得目录下所有文件的路径,并保存到hyatt-k目录下的dir.txt文件
    sortTheDict: 对词典数据进行排序
    getWordList: 获得词表
    getDocName: 获得docID对应的文档相对路径
    """
    def __init__(self):
        self.load_data("./hyatt-k")
        self.invertedIndex = {}
        self.wordList = []
        self.doc_map = IdMap()
        self.wholeDocList = []
        self.dict = {}

    def load_data(self, filePath):
        f = open("./cache/dir.txt", "a")
        for root, dirs, files in os.walk(filePath):
            for file in files:
                f.writelines(os.path.join(root, file).replace('\\', '/') + "\n")
```

Figure 3 Establish 类

索引器封装在 EstablishIndex 类中，主要包含 load_data、sortTheDict、getWordList、getDocName、writeToFile、createIndex 等方法。

```
def sortTheDict(self, dict):
    sortedDict = {k: dict[k] for k in sorted(dict.keys())}
    for stem in sortedDict:
        sortedDict[stem] = {k: sortedDict[stem][k] for k in sorted(sortedDict[stem].keys())}
    return sortedDict
```

Figure 4 sortTheDict 方法

sortTheDict 方法用于对词典进行排序，返回经过排序的词典序列。

```
def getWordList(self):
    # 获得词表
    for word in self.invertedIndex.keys():
        self.wordList.append(word)

def getDocName(self, docID):
    # 返回docID对应的文档相对路径
    return self.doc_map._get_str(docID)
```

Figure 5 getWordList、getDocName 方法

getWordList 方法用于获得词表，getDocName 方法用于返回 docID 对应的文档相对路径:此处使用栈结构存储 docID 和 docName 的映射关系，用 doc_map._get_str 得到 docID 对应的 docName。

```
def writeToFile(self, data, savePath):
    file = open(savePath, 'w')
    str = json.JSONEncoder().encode(data)
    file.write(str)
    file.close()
```

Figure 6 writeToFile 写入文件方法

writeTofile 方法将数据保存为 json 格式。

CreateIndex 方法为索引器的主要方法，读取数据后，进行数据处理，将 term 和 doc 存储到 invertedIndex 列表中，并对倒排索引中的词项进行排序，获得词项列表，最后将数据写入磁盘并保存为 json 格式。

```python
def createIndex(self):
    with open("./cache/dir.txt", "r") as f:
        files = []
        for line in f.readlines():
            line = line.split("\n")
            files.extend(line)
        while '' in files:
            files.remove('')
    os.remove("./cache/dir.txt")
    for file in files:
        print("Analyzing file: ", file)
        doc = PreprocessFile().preProcess(file)  # doc是文档经过预处理后的分词列表
        docID = self.doc_map._get_id(file)
        self.wholeDocList.append(docID)
        num = 0  # term在doc中的位置
        for term in doc:
            if term not in self.invertedIndex:
                docList = {}
                docList[docID] = [num]
                self.invertedIndex[term] = docList
            else:
                if docID not in self.invertedIndex[term]:
                    self.invertedIndex[term][docID] = [num]
                else:
                    self.invertedIndex[term][docID].append(num)
            num += 1
    # 给倒排索引中的词项排序
    self.invertedIndex = self.sortTheDict(self.invertedIndex)
    # 获得词项列表
    self.getWordList()
    # 测试
    # self.printIndex(self.invertedIndex)
    self.dict = dict(zip(self.wholeDocList, files))
    # 数据写入文件
    # self.writeToFile(CompressedPostings.encode(self.invertedIndex), "./cache/invertIndex.json")
    self.writeToFile(self.invertedIndex, "./cache/invertIndex.json")
    self.writeToFile(self.wordList, "./cache/wordList.json")
    self.writeToFile(self.dict, "./cache/doc2docID.json")
    self.writeToFile(sorted(self.wholeDocList), "./cache/wholeDocList.json")
```

Figure 7 createIndex 方法

4.3 文本查询分析器

文本查询主要使用布尔查询方法。用户输入布尔检索表达式后，系统对输入进行检查（拼写校正）并将中序表达式转换为后序表达式（基于栈实现），根据后序表达式在倒排索引表中搜索答案。

布尔检索方法封装在 BoolRetrieval 类中，且由于索引搜索过程中涉及列表的交并差等操作，因此需要调用 operateDocList.py 的各项操作，如

mergeTwoList、andTwoList、listNotcontain。

```python
class BoolRetrieval:
    def __init__(self):
        pass
    def boolOperator(self, oper):
        # 布尔检索操作数
        precedence = ['OR', 'AND', 'NOT']
        for i in range(3):
            if oper == precedence[i]:
                return i
        return -1
```

Figure 8 BookRetrieval 类

将用户输入的布尔表达式（中序表达式）转换为后序表达式，使用栈实现此操作。

```python
def inf2Profix(self, inputList):
    # 中序表达式转换为后序表达式，将操作符放在表达式后
    precedence = {}
    precedence['OR'] = 0
    precedence['AND'] = 1
    precedence['NOT'] = 2
    profix_res = []
    temp = []
    queries = []
    for word in inputList:
        if word == '(':
            temp.append('(')
        elif word == ')':
            if len(queries) > 0:
                profix_res.append(queries)
                queries = []
            sym = temp.pop()
            while sym != '(':
                profix_res.append(sym)
                if len(temp) == 0:
                    print("Incorrect Query")
                    exit(1)
                    break
                sym = temp.pop()
```

Figure 9 inf2profix 方法

```
        elif word == 'NOT' or word == "OR" or word == 'AND':
            if len(queries) > 0:
                profix_res.append(queries)
                queries = []
            if (len(temp) <= 0):
                temp.append(word)
            else:
                sym = temp[len(temp) - 1]
                # 弹出到左括号为止
                while len(temp) > 0 and sym != '(' and precedence[sym] >= precedence[word]:
                    # pop out
                    profix_res.append(temp.pop())
                    if (len(temp) == 0):
                        break
                    sym = temp[len(temp) - 1]
                # push in
                temp.append(word)
        else:
            queries.append(word)
    if len(queries) > 0:
        profix_res.append(queries)
    while len(temp) > 0:
        profix_res.append(temp.pop())
    return profix_res
```

Figure 9 inf2profix 方法（续）

Query_to_search 方法根据用户输入的布尔表达式对倒排索引表进行搜索返回对应的文档结果，其实现如下。

```
def query_to_search(self, query, index):
    pofix = self.inf2Profix(query)
    result = []
    print(pofix)
    # queryArray = []
    # notTrue = ['1']
    # notFalse = ['0']
    nullReturn = []
    limit = len(pofix)
    i = 0
    while i < limit:
        item = pofix[i]
        if item != 'AND' and item != 'OR':
            if i < limit - 1:
                if pofix[i + 1] == "NOT":
                    i = i + 1
                    result.append(self.serarchPhraseForBool(index, item, flag=False))
                else:
                    result.append(self.serarchPhraseForBool(index, item, flag=True))
            else:
                result.append(self.serarchPhraseForBool(index, item, flag=False))
```

Figure 10 query_to_search 方法

```
    elif item == 'AND':
        if len(result) < 2:
            print("illegal query")
            return nullReturn
        else:
            list1 = result.pop()
            list2 = result.pop()
            result.append(andTwoList(list1, list2))
    elif item == 'OR':
        if len(result) < 2:
            print("illegal query")
            return nullReturn
        else:
            list1 = result.pop()
            list2 = result.pop()
            result.append(mergeTwoList(list1, list2))
    i += 1
if len(result) != 1:
    print("illegal query")
    return nullReturn
else:
    return result.pop()
```

Figure 10 query_to_search 方法（续）

4.4 拼写校对

使用基于字符串的编辑距离进行拼写校正（动态规划算法），使用 Gutenberg 数据集作为语料库。

```
class SpellingCheck:
    def __init__(self):
        # Corpus: Big.txt: http://norvig.com/big.txt
        # Gutenberg语料库数据，维基词典，英国国家语料库中最常用单词列表
        with open('cache/big.txt', 'r') as f:
            WORDS = self.tokens(f.read())

        self.WORD_COUNTS = collections.Counter(WORDS)

    # 基于字符串的编辑距离进行拼写校正(动态规划算法)
    # Or NLTK模块中edit_distance()函数
    def tokens(self, text):
        """
        Get all words from the corpus
        """
        return re.findall('[a-z]+', text.lower())
```

Figure 11 SpellingCheck 类

使用 edits0、edits1、edits2 返回与输入词相差 0、1、2 个编辑的字符串的所有字符串；使用 correct 方法返回正确拼写，优先级为编辑距离 0、编辑距离 1、编辑距离 2。

```
    def known(self, words):
        # 返回实际在我们的WORD_COUNTS字典中的单词子集
        return {w for w in words if w in self.WORD_COUNTS}

    def edits0(self, word):
        # 返回所有与输入词相差0个编辑的字符串的所有字符串
        return {word}

    def edits1(self, word):
        # 返回所有与输入词相差1个编辑的字符串的所有字符串
        alphabet = ''.join([chr(ord('a') + i) for i in range(26)])

        def splits(word):
            return [(word[:i], word[i:])
                    for i in range(len(word) + 1)]

        pairs = splits(word)
        deletes = [a + b[1:] for (a, b) in pairs if b]
        transposes = [a + b[1] + b[0] + b[2:] for (a, b) in pairs if len(b) > 1]
        replaces = [a + c + b[1:] for (a, b) in pairs for c in alphabet if b]
        inserts = [a + c + b for (a, b) in pairs for c in alphabet]
        return set(deletes + transposes + replaces + inserts)

    def edits2(self, word):
        # 返回所有与输入词相差2个编辑的字符串的所有字符串
        return {e2 for e1 in self.edits1(word) for e2 in self.edits1(e1)}
```

Figure 12 编辑距离算法

```
def correct(self, word):
    # 返回正确拼写
    # 优先级为编辑距离0，然后是1，然后是2
    candidates = (self.known(self.edits0(word)) or
                  self.known(self.edits1(word)) or
                  self.known(self.edits2(word)) or
                  {word})

    return max(candidates, key=self.WORD_COUNTS.get)

def correctMatch(self, match):
    word = match.group()

    def case_of(text):
        return (str.upper if text.isupper() else
                str.lower if text.islower() else
                str.title if text.istitle() else
                str)

    return case_of(word)(self.correct(word.lower()))

def spellingCheck(self, text):
    return re.sub('[a-zA-Z]+', self.correctMatch, text)

def reSub(self, text):
    remove_chars = '[0-9’!"#$%&\'()*+,-./:;<=>?@，。?★、…【】《》？“”‘’！[\\]^_`{|}~]+'
    return re.sub(remove_chars, '', text)
```

Figure 13 correct、correctMatch、spellingCheck 方法

4.5 索引压缩器

使用 vb 编码对索引进行压缩。将索引过程封装在 CompressedPostings 类中。

```
class CompressedPostings:

    @staticmethod
    def vbcode(n):
        # vb编码：一种可变长的字节编码
        byte = []
        while True:
            byte.append(n % 128)
            if n < 128:
                break
            n = n // 128
        byte[0] += 128
        byte = list(reversed(byte))
        return byte
```

Figure 14 Compressed Postings 类

encode 方法使用 gap-encoding 压缩差值 gap＝posting－last，对索引进行压缩。

```
@staticmethod
def encode(postings_list):
    # gap-encoding 压缩差值gap = posting - last
    bytestream = []
    last = 0
    for posting in postings_list:
        gap = posting - last
        last = posting
        byte = CompressedPostings.vbcode(gap)
        bytestream.extend(byte)
    return array.array('B', bytestream).tobytes()
```

Figure 15 encode 方法

decode 方法用于对索引进行解码。

```python
@staticmethod
def decode(encoded_postings_list):
    decoded_postings_list = array.array('B')
    decoded_postings_list.frombytes(encoded_postings_list)
    numbers = []
    n = 0
    for i, byte in enumerate(decoded_postings_list):
        if byte < 128:
            n = 128 * n + byte
        else:
            n = 128 * n + byte - 128
            numbers.append(n)
            n = 0
    prefix_sum = 0
    res = []
    for num in numbers:
        prefix_sum += num
        res.append(prefix_sum)
    return res
```

Figure 16 decode 方法

（五） 测试

布尔检索测试：（分别测试 AND、OR、NOT 等操作符）

```python
# input = ["(", "americn", "AND", "china", ")"]
input = ["(", "americn", "OR", "china", ")"]
input = ["(", "americn", "AND", "china", ")"]
input = ["americn", "AND", "china"]
input = ["NOT", "english", "AND", "china"]

# 拼写校正
correct_input = [SpellingCheck().spellingCheck(i) for i in input]
result = BoolRetrieval().query_to_search(correct_input, invertedIndex)
# docID转换为doc目录
re_result = [doc2docID[str(i)] for i in result]
print(re_result)
```

Figure 17 布尔检索测试

```
[['english'], 'NOT', ['china'], 'AND']
['./hyatt-k/deleted_items/119', './hyatt-k/deleted_items/124', './hyatt-k/deleted_items/129', './hyatt-k/deleted_items/211', '
```

Figure 18 测试结果

测试阶段基于 easygui 设计一简单交互界面如下：

1. 首先用户点击输出按钮，输入您要检索的内容。（此界面为一交互启动界面，并无实际交互操作。）

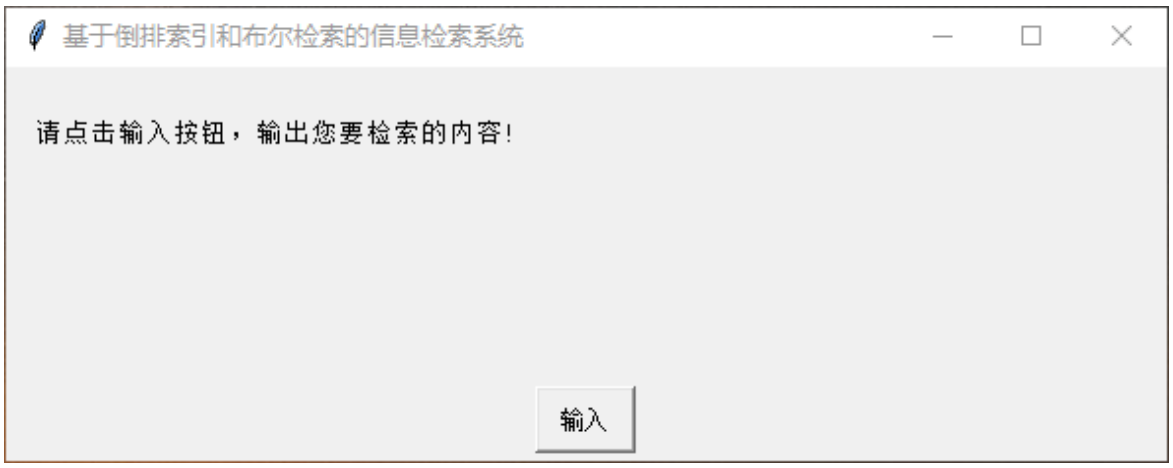

Figure 19 界面 1

2. 用户需要选择是否需要进行拼写纠正：若选择不需要，则在布尔检索时不会调用拼写检查功能（此需求适用于对检索结果有更高精度要求的用户）；若需要，则在布尔检索时将调用拼写检查功能，即自动对用户输入进行检错纠错处理（可能对于某些非常见的检索会出现错误判断）。

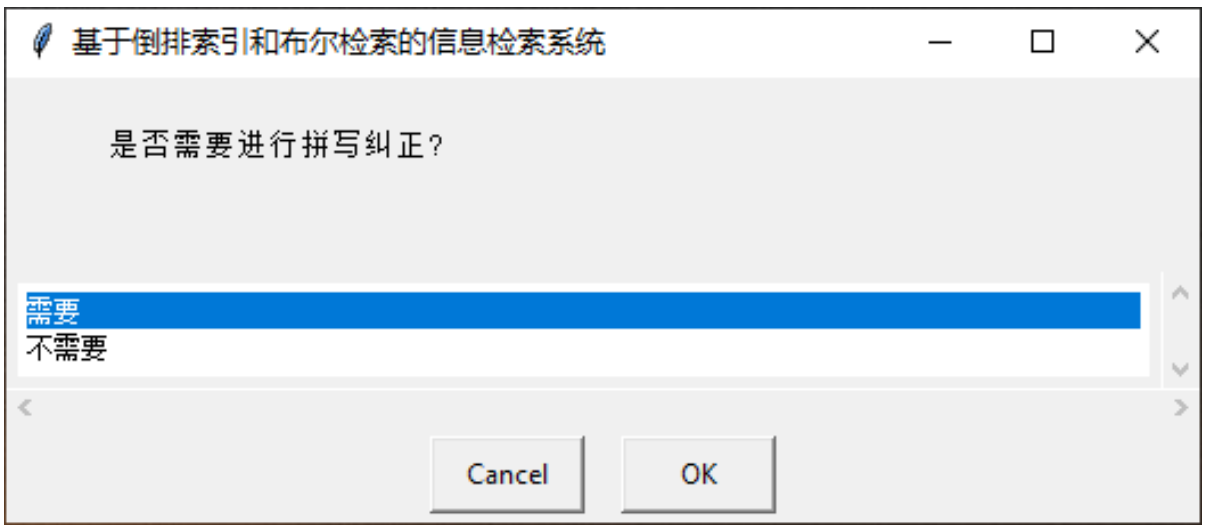

Figure 20 界面 2

3. 输入您需要检索的内容，要求按照布尔检索表达式的格式。

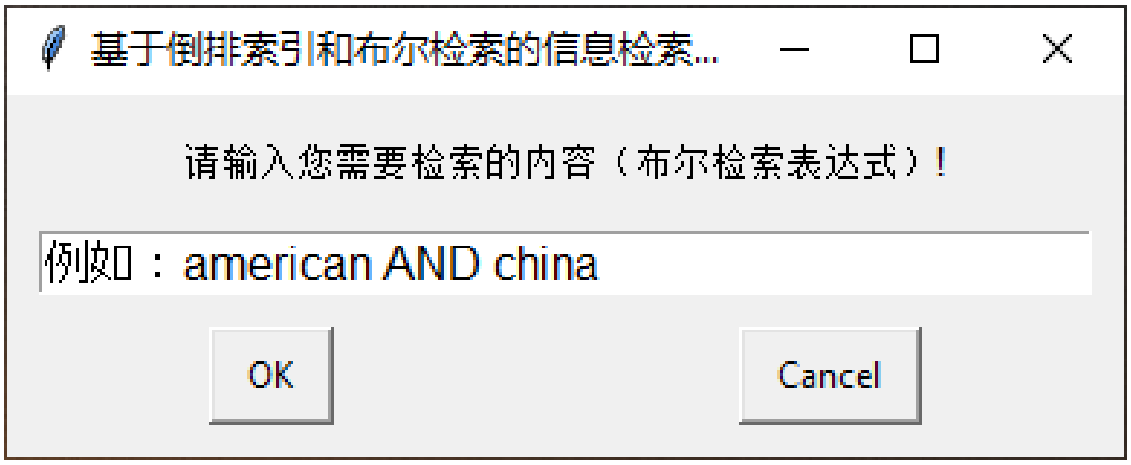

Figure 21 界面 3

4. 假设采用默认输入 american AND china，则输出结果如下。(返回对应文档路径)

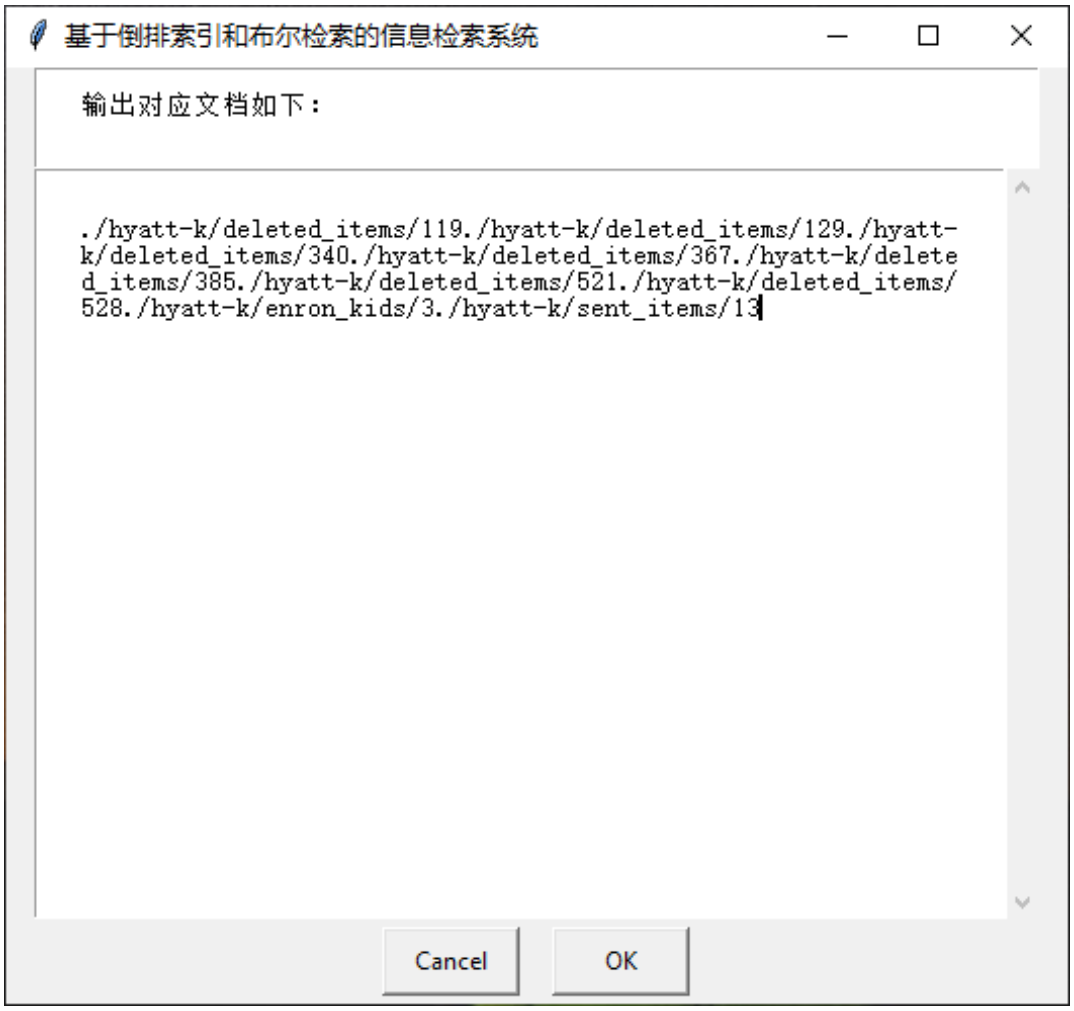

Figure 22 界面 4

（六） 注意事项与项目思考

1. 注意事项：运行时需要先运行 establishIndex.py 得到倒排索引表和词项表并存储到./cache 文件下。

2. 项目思考：由于实验样本相对较小，对于系统可处理数据的规模及操作时间要求相对较低，实际操作中应该更加注重这一部分。